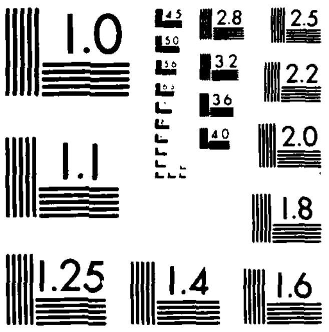
AD-A184 243 AUTONOMOUS LAND VEHICLE (ALV) PLANNING AND NAVIGATION 1/1
SYSTEM(U) HUGHES RESEARCH LABS MALIBU CA
D M KEIRSEY ET AL 06 APR 87 ETL-0465 DACA76-85-C-0017
UNCLASSIFIED F/G 13/6 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3

ETL-0465

Autonomous Land Vehicle (ALV) Planning and Navigation System

D. M. Keirsey
J. S. B. Mitchell
D. W. Payton
D. Y. Tseng
V. S. Wong
K. Zikan

Hughes Research Laboratories
3011 Malibu Canyon Road
Malibu, CA 90265

April 1987

Prepared for
U.S. ARMY CORPS OF ENGINEERS
ENGINEER TOPOGRAPHIC LABORATORIES
FORT BELVOIR, VIRGINIA 22060-5546

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

A1

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) DACA76-85-C-0017	5. MONITORING ORGANIZATION REPORT NUMBER(S) ETL-0465
6a. NAME OF PERFORMING ORGANIZATION Hughes Research Laboratories; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION U.S. Army Engineer Topographic Laboratories
6c. ADDRESS (City, State, and ZIP Code) 3011 Malibu Canyon Road Malibu, CA 90265	7b. ADDRESS (City, State, and ZIP Code) Fort Belvoir, VA 22060-5546
8a. NAME OF FUNDING/SPONSORING ORGANIZATION U.S. Army Engineer Topographic Labs; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Fort Belvoir, VA 22060-5546	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO.; TASK NO.; WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification) ALV Planning and Navigation System

12. PERSONAL AUTHOR(S) D.M. Keirsey, J.S.B. Mitchell, D.W. Payton, D.Y. Tseng, V.S. Wong, and K. Zikan

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Year~~ly~~ Technical 1	FROM 9/85 TO 10/86	87 April 6	72

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	autonomous vehicles; terrain navigation; route planning; mobile robots; hierarchical planning

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Hughes planning system is designed to achieve the reasoning requirements of the Autonomous Land Vehicle (ALV) for navigation in unconstrained outdoor environments. This system is designed specifically to handle diverse terrain with maximal speed, efficacy and versatility. The hierarchical architecture for this system is presented along with the detailed algorithms, heuristics, and planning methodologies for the component modules. The architecture is structured such that lower-level modules perform tasks requiring greatest immediacy while higher-level modules perform tasks involving greater assimilation of sensor data, making use of large amounts of a priori knowledge.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED [✓ SAME AS RPT] ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL D.Y. Tseng	22b. TELEPHONE (Include Area Code) (818) 702-5267	22c. OFFICE SYMBOL

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE Unclassified

This report documents the first year of progress under contract to DARPA for the Planning and Navigation System for ALV. It includes a discussion of the technical design of the multi-level planning system and its component modules. Specific details of progress made in developing map-based planning capabilities for the Martin Marietta test area, simulation capabilities to support development of real-time sensor-based motion planning, and several reflexive behaviors and virtual sensors which provide basic road following and obstacle avoidance capabilities.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1 INTRODUCTION

This is the annual report for the Hughes ALV Planning and Navigation System program (Contract #DACA76-85-C-0017) describing the results of the technical efforts for the period September 19,1985 to September 19,1986. This program is part of the DARPA Strategic Computing Program. Dr. Isler. DARPA/ISTO, is the program manager, and Rose Holecheck is the COTR.

The Hughes system is designed to achieve the planning goals of the Autonomous Land Vehicle (ALV) in an unconstrained outdoor environment. A reasoning system to support planning and control requirements of the ALV will be described. This system is designed specifically to handle diverse terrain with maximal speed, efficacy, and versatility. For a system with the complexity of the ALV, conventional methods of system design involving distinct functional modules with dedicated processes no longer suffice. Functions and processes are intimately interrated, and the interactions between the major sub-systems of the ALV (such as perception, planning, and action) are frequent and dynamic. For this reason, we believe that the Hughes multi-level system architecture (developed under IR&D) is ideal for achieving the functional and processing requirements of the ALV. We have the unique opportunity to apply this architecture in both the planning and perception components of the ALV through our close interaction with individuals working on a parallel effort at Hughes for Knowledge Based Vision Technology (KBVT).

The hierarchical architecture for this system is presented along with the detailed algorithms, heuristics, and planning methodologies for component modules. The architecture is structured such that lower-level modules perform tasks requiring greatest immediacy while higher-level modules perform tasks involving greater assimilation of sensor data, making use of large amounts of a priori knowledge. In describing the component modules of this system, specific techniques for mission planning, map-based route planning, local terrain navigation, and reflexive vehicle control are presented. These techniques have been demonstrated both in a detailed real-time simulation and on a small indoor robotic vehicle. Next year's results will include demonstrations on Martin's ALV. (Tests have been performed on the ALV, but not in the time period covered by this report.)

1.1 THE PROBLEMS

The objective of operating in unconstrained natural environments poses many problems for the perception and planning components of an autonomous system. The complexity of the environment makes it impossible for the planning component to have complete a priori knowledge of every relevant detail necessary to generate and execute a plan. This, in turn, places the burden on the perception component to supply those details, but the lack of constraints also makes recognition tasks very difficult. Even if we presume to have a perception component which can recognize critical features in the environment, the value of these observed features rapidly decays with vehicle motion and the passage of time. Vehicle motion is constantly exposing new features and details of the environment which will most likely be more important than those already observed. Meanwhile, as objects other than the vehicle move in the environment, past observations of those objects become meaningless.

When we think of the planning problems that humans must solve in performing traversal tasks through unfamiliar terrain. we tend to arrive most readily at problems which involve a high degree of cognitive thought. We quickly recognize the need for skills such as interpreting and understanding a mission request. using a map to select an efficient route, and navigating with a map and possibly a compass to be sure we stay on a planned route. At the same time. however, there are many planning problems which we tend to take for granted. having learned to solve them so well that we no longer have to think about them. Tasks such as driving down a road, turning at intersections, and avoiding obstacles at one time required learning and practice for us to become sufficiently proficient to execute them properly. For an autonomous vehicle. tasks such as these may be far more difficult to implement than those which require a higher degree of conscious effort.

When we consider the perception problems relating to mobility, it is clear that the majority of human perception tasks are performed at a subcognitive level. Personal experience tells us that we can just look in any direction and our minds are immediately flooded with a multitude of recognized features and objects from the environment. We must be careful, however, not to base our assumptions about perception for an autonomous vehicle on our conception of human perceptual abilities. The technology for machine perception is still a long way from the levels of sophistication that we find in higher mammals. We cannot ask a machine perception system to describe the environment without first indicating what it is expected to see. Even then, we will be fortunate if a machine perception system can accurately detect all of the features we might deem important for controlling a vehicle.

If we are to avoid posing the autonomous vehicle problem as purely a perception problem, the vehicle planning system must make maximal use of whatever knowledge it may have at its access to help direct perception tasks. Using available map data and knowledge of its near-term goals, the planning system must be able to direct the attention of its available perception resources so that it can get the information it requires to move through the environment. Knowing that the vehicle is on a road, for example, should enable the planning system to constrain the perception system to provide nothing more than a description of the road boundaries and obstacles on the road, thereby relieving the perception system from the awesome task of describing all that is visible to it.

1.2 THE SOLUTIONS

This report highlights the progress for our first year under contract to DARPA for the ALV program. To provide the background for this work, we also present a summary of previous research performed under an Army contract for Intelligent Tactical Autonomous Control (ITAC), and IR&D. Although many problems remain to be solved before the ultimate goal of autonomous vehicle mobility may be achieved, we feel this research has made contributions in addressing serveral specific planning domains and in providing an architecture into which these planning techniques may be embedded. The planning approaches presented herein were developed initially as independent solutions to distinct subproblems in autonomous vehicle planning. As they evolved, however, it was found that the logical basis for maintaining the distinction between these approaches was quite compelling. Moreover, these distinctions eventually led to the development of more general architectural concepts which have greatly clarified the task domain boundaries for the component planning units and have helped to define the interfaces between them.

2 SYSTEM ARCHITECTURE

2.1 OVERVIEW

Our overall autonomous vehicle system architecture is decomposed into three primary subsystems: a planning system, a perception system, and a system monitor. As Figure 1, illustrates, all sensory input flows through the perception system and, at varying levels of abstraction, flows from the perception system to the planning system. The planning system then has the responsibility to react to this information and evoke vehicle actions. Meanwhile, the system monitor has the ability to monitor activity within both the perception and the planning systems, altering the default operating constraints to compensate for any detected failures.

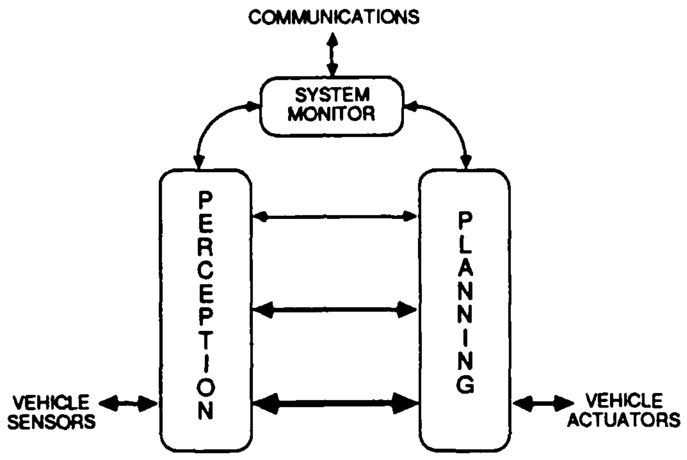


Figure 1. Autonomous Vehicle Architecture

Control within the overall system operates on the basis of loosely coupled master-slave relationships between the three independently controlled cooperating systems. There is no need for a central controlling agent within this system since each of the three subsystems has its own well-defined role and modes of interaction. The role of the planning system is to reason about mission objectives and produce appropriate vehicle actions. In performing this role, it must issue requests to the perception system, which is responsible for providing suitable descriptions of all requested environmental features relevant to the production of vehicle actions. The system monitor has influence over the perception and planning systems only to the extent that it can constrain their modes of operation. As each of these systems must perform a number of distinct problem solving functions, they must each be regarded as independent agents placed within a framework for cooperation.

Before describing the internal architecture of each of the component systems, it will be helpful to clarify their functional and performance requirements, and highlight the various design tradeoffs that were taken into consderation as the architecture evolved. After providing an overall view of the functions and structure of the planning system, perception system, and system monitor, the remainder of this report will focus on details of the planning system architecture and its component algorithms.

2.2 FUNCTIONAL REQUIREMENTS

Although many detailed aspects of planning, perception, and system monitoring are tightly intertwined, it is possible, at least at an abstract level, to recognize how various tasks may fit into each of these categories.

By understanding the specific tasks that are to be performed within these categories, we can arrive at meaningful module subdivisions and obtain a better idea of what interfaces will be required.

2.2.1 Planning Tasks

The role of planning is to translate abstract mission goals into concrete physical vehicle actions. Clearly, a wide range of tasks are required to make this translation possible. First, with a mission specification provided, mission analysis must be performed to determine appropriate intermediate goals and route constraints. Then, map data must be analyzed to obtain detailed routes through known terrain. Appropriate control actions must then be selected to follow the intended routes. Although the best effort is made to plan routes which avoid threats and undesirable terrain conditions, it is inevitable for an autonomous vehicle in a complex dynamic environment to encounter unexpected or unavoidable adversity. When these situations arise, a capability will be needed for varying levels of plan revision or replanning depending on the nature of the plan failure.

2.2.2 Perception Tasks

The primary role of perception is to support all of the sensory needs of planning. This includes the acquisition of sensory data as well as the assimilation of acquired data. Since planning involves such a wide range of tasks and capabilities, it is inevitable that perception must support a comparably wide range of tasks and capabilities. The use of these capabilities, however, is focused through requests for information issued by the planning system. Although planning requests serve as an attention-focusing mechanism, perception must be capable of satisfying multiple requests simultaneously. Since different requests may demand different forms of data with different requirements for assimilation or immediacy of that data, perception must be capable of fusing data from multiple sensors as well as providing specialized processing for various real-time control tasks.

2.2.3 System Monitor Tasks

The role of the system monitor is to serve as an overall system watchdog, record-keeper, and communications manager. Its key monitoring functions include detection, diagnosis, and correction of system malfunctions. Although the perception and planning components both have their own localized failure detection capabilities, the system monitor is needed to detect inconsistencies in the operation of the composite system, checking the sanity of normally disassociated components which might inadvertantly evoke disparate actions. As the overseer of system operations, it is natural for the system monitor to maintain a record of significant events occuring in either planning or perception. This historical event log will support both operational needs such as detailed failure diagnosis, and base-oriented needs such as maintenance support and post-operational performance assessment. As communications manager, the system monitor will provide a centralized interface to the outside world. Using its diagnostic capabilities, the system monitor will be able to provide succinct reports of status and failure conditions, and possibly to obtain further instructions from human operators when necessary.

2.3 PERFORMANCE REQUIREMENTS

An autonomous vehicle must be capable of traveling quickly and efficiently through diverse environmental conditions while attaining a variety of possible mission objectives. Clearly, the complexity of this task requires fairly sophisticated sensing and reasoning capabilities, but such capabilities are useless if they cannot be made to satisfy the real-time needs of a vehicle in a complex dynamic environment. Not only must the system be capable of implementing a wide variety of operational strategies, but it must readily be able to change its strategies as environmental conditions change.

A system that can intelligently control an autonomous vehicle within a complex dynamic environment must have the responsiveness to react quickly to constant changes in the environment and the cunning to operate with a rich repertoire of strategies and tactics. These performance requirements will have a significant influence on all aspects of the detailed system design.

2.3.1 Short Reaction Time

With continuous vehicle motion, sensor data quickly becomes obsolete, so the time between receiving data and acting on it is critical. The prospect of continuous vehicle motion places serious constraints on the time available for sensor data processing and plan generation. As soon as sensor data is acquired, vehicle motion begins to make that data obsolete for planning. The farther the vehicle has moved from the point where data was obtained, the less useful that data becomes. Some data may lose value more quickly than other data. For example, a perception report of a nearby threat may only be useful if that report can be processed and acted upon before a weapon is launched, while a report of a distant hill or mountain may remain useful for quite some time. In certain instances, data becomes obsolete when different vantage points expose new aspects of the environment that were previously occluded. For example, a perception report of a clear area ahead may quickly become invalidated when the vehicle passes over a hill and into the visibility range of enemy radar. The problem of data obsolescence must be minimized through timely processing of sensor data and use of efficient methods for plan verification and updating.

2.4 REPERTOIRE OF STRATEGIES

Diverse terrain and adversary threats pose another challenge to planning in that different environmental conditions may require the use of different strategies for effective mobility. A wide variety of planning strategies may also require a variety of sensing capabilities for their support. These requirements may lead to overwhelming computational demands if appropriate tradeoffs between immediacy and assimilation are not made.

Traveling down a road may involve considerably different planning skills and sensing capabilities than traveling cross-country. The primary concerns when following a road are to travel quickly while staying on the road and avoiding obstacles. This is unlike negotiating undeveloped terrain where, in addition to obstacle avoidance, the concerns range from climbing steep slopes to avoiding stoppage by potholes, gullies, and swamps. Different strategies may be needed in these situations to make use of specialized combinations of sensors, and to perform tasks that are specialized to the environmental conditions.

A well diversified set of task capabilities and strategies may also prove useful when adversary threat situations are introduced to the vehicle environment. Under these circumstances, predictable behavior is highly undesirable, as it provides an adversary with advantageous knowledge of one's weaknesses. With the availability of more than one method for accomplishing most tasks, the vehicle planner should be able to vary its strategies in such a way as to reduce its own predictability.

The variety of sensing capabilities needed to support different planning strategies leads to some serious sensor fusion problems. Many types of sensors will all be acquiring data about the environment. Although various sensors will be especially useful for obtaining data about particular attributes of the environment, a great deal of useful information will also be found in the interrelationships between outputs of different sensors. The task of analyzing data from all sensors and constructing a composite view of the world, however, may be incompatible with the need for timely information about the environment. At some point, the benefits of data assimilation must be sacrificed in order to achieve the immediacy needed for real-time response.

2.5 DESIGN TRADEOFFS

A number of design tradeoffs have had a critical influence in the organization and structure of our system architecture. Key concerns revolve around the need to obtain the benefits of thorough sensory data assimilation without sacrificing the ability to react immediately to partially assimilated data. Two alternative approaches to system decomposition have been considered which differ significantly in their potential impact on our achievement of this goal. Not only do these approaches vary in terms of providing the desired combination of immediacy and assimilation, but they also result in extremely different data flow patterns and development methodologies. Ultimately, we have selected a vertical decomposition approach because it can provide the advantages of a spectrum of immediacy and assimilation tradeoffs while also establishing a structure which promotes a highly modular development methodology.

2.5.1 Immediacy Versus Assimilation

In making the tradeoff between immediacy and assimilation, there is a wide spectrum of options. There are certain advantages and disadvantages at each extreme which are also evident in differing strengths anywhere along this spectrum.

2.5.1.1 Assimilation.

On the positive side of assimilation is that the completeness and detail of a constructed world representation can always be enhanced with added processing of sensor data. This clearly has value to planning, since features critical to successful plan execution may not otherwise be discovered. The negative aspects of assimilation come as a result of the time required to obtain it. As a sample of sensor data is undergoing painstaking analysis to detect potential obstacles, the vehicle may have already run into one of them.

As demonstrated by work in several autonomous vehicle projects, [Crowley85], [Moravec83], [Nilsson84] thorough sensor processing may allow construction of a scene representation which extends well into the vehicle's projected path, thereby compensating for the time required to construct such a representation. Despite the fact that a portion of the model may become obsolete due to vehicle motion, there are still enough valid parts of the model remaining to properly control the vehicle. Even these projects, however, must face some requirements for immediacy, invoking action as soon as only the most minimal models for their environmental context have been constructed.

2.5.1.2 Immediacy.

On the positive side of immediacy is the fact that the faster any sensory data can be used to effect action, the more value it has for control. The combined vehicle perception and planning systems may be considered as a single feedback control system with the environment providing the feedback signal to that system. Since greater stability in a feedback control system is generally achieved by reduction of delays through the system, greater immediacy should simplify the task of maintaining stable control. This simplification may in turn make further reduction in delay possible. Greater immediacy also has value in reducing the likelihood that unexpected changes in the environment may occur between successive data inputs, since these inputs are received more frequently. The problem with immediacy is that critical information may be difficult to obtain from sensory data that has not undergone sufficient assimilation, and it is likely that some extracted information may contain errors or inconsistencies.

Several systems have been developed which use control strategies that are designed specifically to take advantage of data immediacy whenever possible [Brooks85], [Harmon84], [Wallace85]. These systems have shown that immediate data can be effective in well constrained situations for which there is sufficient a priori knowledge to allow simple disambiguation of sensor data. Data from acoustic range finders, for example, can be adequate for control with very little processing, provided that the environment does not contain hazards which are undetectable to these sensors. Because immediate data loses value in loosely constrained environments, systems exploiting immediacy tend either to limit operation to well constrained environments or to employ complementary methods involving higher degrees of assimilation to establish the needed constraints.

2.5.2 Vertical Versus Horizontal System Decomposition

It is possible to view the modularization of the autonomous vehicle problem as one with either a horizontal or a vertical decomposition [Brooks85]. In the more traditional horizontal decomposition approach [Crowley85], [Moravec83], [Nilsson84], information flows through a series of assimilation stages, then through a series of planning stages, until finally it results in the issuance of a control signal. Each successive stage in the pipeline requires assimilated data from the preceding stage as its input. In the vertical decomposition approach [Brooks85], information is dispersed to a set of parallel layers. Each specialized layer processes data in a manner specific to its own operational goals, and issues control commands accordingly. Although higher level layers are expected to exert influence upon lower level layers, the lower layers are free to exert control over the vehicle without waiting for assimilation to be completed by the upper levels. These two approaches to system decomposition differ markedly in the resultant data flow and system development methodology.

2.5.2.1 Horizontal System Decomposition

Data Flow. In the horizontal approach to system decomposition, the fusion of data from multiple sensors is generally performed through the generation of a composite representation of the local environment. A series of sensor assimilation and integration stages are typically used to produce the representation, which provides subsequent planning stages with data for vehicle control. Such representations are desirable because they can be meaningful over a broad range of situations, and they may allow for fairly general planning approaches to function effectively under a variety of environmental conditions.

Although the generality of a composite representation has its advantages, it also has some serious disadvantages. The primary drawback is that the series of stages through which all sensor data must pass places an unavoidable delay in the loop between sensing and action. Unless the composite representation is bypassed at times, thereby undermining the generality of the approach, the delay can be reduced only by increasing the throughput of some or all of the modules. At times it may occur that specialized circumstances or goals provide sufficient constraints to allow direct disambiguation of sensor data for control, but the specialized functions needed to make this possible may not be easily added if the architecture has not already been organized to accommodate them.

Development Methodology. A horizontal decomposition presents some serious problems to system development as well, because of limitations imposed by the need for early interface specification. As a system is partitioned into modules, a set of interfaces must be defined to establish information pathways between modules. These interfaces are important since they specify both the information content and the format for transferred data. The specification of any module, however, depends heavily on the specification of its inputs and outputs, so modules cannot be designed without first obtaining a stable interface specification. As a result, interface specification occurs early in the design cycle of the system, placing serious constraints on immediacy and assimilation requirements before the potential capabilities of each of the modules are completely understood. Development of each of the modules is thus restricted by the interface limitations, discouraging otherwise desirable module enhancements if they cannot be structured to conform to these specifications. Furthermore, should a particular enhancement be found sufficiently useful to warrant modification of an interface, effecting such a change might require significant redesign of the system. At the very least, the modules on each side of the altered interface would need to be changed, and it is likely that these changes would result in the need to change other interfaces and modules as well. To avoid this difficulty, emphasis is often placed on maximizing the throughput of component modules rather than enhancing their functional capabilities. Faster hardware is often seen as the only practical solution to problems which might otherwise be solved with alternate module functionality were it not for the interface constraints.

2.5.2.2 Vertical System Decomposition.

Data Flow. In the vertical approach to system decomposition, multiple sensor data may be fused through vehicle actions as well as within internal composite representations. A parallel set of sensor assimilation and fusion tasks are typically performed, leading to the generation of a variety of specialized representations for use by a number of concurrent planning tasks. As each planning task employs representations that may be derived from only a small portion of the available sensor data, many distinct, and possibly conflicting, control commands may result. This opens the possibility of bypassing sensor fusion in exchange for command fusion.

Although the use of command fusion lacks the generality obtained from full sensor fusion, it has some definite advantages. Primarily, it allows pathways for primitive sensor data to be exploited for immediate vehicle response. For such specialized pathways to be used effectively, they must be governed on the basis of more thoroughly assimilated data to maintain proper constraints for meaningful sensor disambiguation. This capacity, however, is easily provided from parallel assimilation and planning tasks.

Development Methodology. Under the vertical decomposition approach, development is organized around a hierarchical set of layers where each layer may be influenced by its superiors but is independent from them in performing its own tasks. In this kind of hierarchy, the lower level modules perform more simple and general tasks while the higher level modules perform more complex and situation specific tasks.

The modules of a vertical decomposition may be fairly independent and simple, allowing easy modification to accommodate new approaches and solutions as they arise. With each module having its own specialization and operating goals, each may also have its own specialized interfaces. This delegates the problem of interface specification more appropriately as a subtask of module design. Interfaces can thus be designed according to the unique requirements of each module while remaining sufficiently independent to allow changes to a module's functionality with only a minor impact on other modules.

The added flexibility provided by the vertical decomposition approach allows more effective tradeoffs to be made between immediacy and assimilation. Because the design involves a number of special-purpose interfaces rather than a few very general interfaces, a larger variety of different immediacy/assimilation tradeoffs may be made for the overall system. With a different tradeoff selected appropriately for each level of the system hierarchy, the total system may effectively take advantage of a wide range of immediacy/assimilation options simultaneously. Furthermore, this coverage of the immediacy/assimilation spectrum may be changed easily as the system develops and interfaces are modified.

2.6 THE HIERARCHICAL ARCHITECTURE

The system is structured as a distributed control hierarchy in which lower level modules perform tasks requiring greatest immediacy while higher-level modules perform tasks involving greater assimilation of sensor data. Similar hierarchical architectures have been presented previously ([Chavez84], [Isik86], [Giralt79]). Since each layer of the hierarchy includes both a perception component and a planning component, the system may be viewed as parallel planning and perception hierarchies which communicate at a number of different levels of abstraction. With all modules of the hierarchy operating in parallel, higher-level planning modules perform selective activation and deactivation of lower-level responses while higher-levels of perception assimilate data from their lower levels. The result is that data assimilation and complex reasoning can occur with limited interference from demands for immediacy, allowing the system to exploit the advantages of both immediate and assimilated data.

Figure 2 shows the detailed system architecture. As mentioned earlier, the system is decomposed into three primary components: a perception system, a planning system, and a system monitor. Perception and planning are structured as hierarchies with distinct interfaces at each level. Each module of the planning and perception hierarchies interacts only with its immediate subordinate and superior modules within the same hierarchy, and only with its corresponding module in the companion hierarchy. In contrast, the system monitor has access to every level of both the perception and the planning hierarchies, allowing it to obtain a composite picture of system operation and to impose new operating constraints at any level.

The horizontal and vertical interface lines in Figure 2 illustrate different pathways of information flow within the system and reflect differences in information content as well. Within the vertical interface pathways, control, constraints, and requests are passed downward to subordinate hierarchy layers while status, failure, and responses are passed upward to superior hierarchy layers. Within the horizontal interface pathways, requests for data flow from planning to perception while assimilated data flows from perception to planning. As illustrated by the thickness of these interface lines, lower levels of the hierarchy require higher bandwidth interfaces to support the real-time aspects of vehicle control.

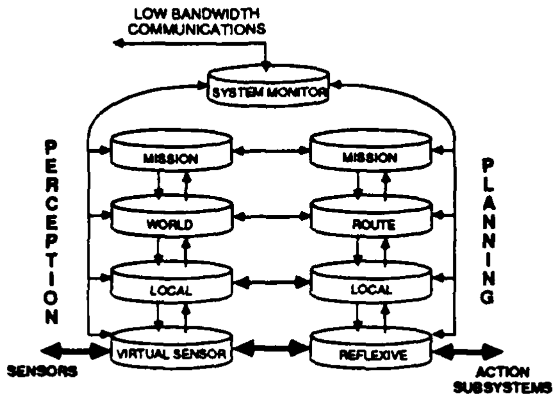


Figure 2. Overall System Architecture

2.7 THE PLANNING SYSTEM ARCHITECTURE

2.7.1 Overview

Because the primary objective of our research has been to develop a planning system for autonomous vehicles, the remainder of this paper will focus on the planning architecture and its component algorithms. In developing our architecture, we have attempted to address all aspects of autonomous vehicle planning, ranging from mission analysis to executing routes. In addition, we have attempted to specify how each aspect of planning might interact with corresponding perceptual resources such that all perceptual tasks can be clearly directed and specialized in order to efficiently produce well-defined responses. The resulting multilevel planning system is structured as a four-level hierarchy as shown in Figure 3. The distinction between different levels of this hierarchy is based primarily on the need for different degrees of immediacy or assimilation within the system. The major modules in this hierarchy are intended to be operated in parallel, with modules near the top performing tasks requiring a high degree of assimilation and modules near the bottom performing tasks requiring a high degree of immediacy. In this way, the hierarchy covers the entire spectrum of immediacy/assimilation tradeoffs, deriving unique benefits from each level.

2.7.2 System Decomposition

In this vertical decomposition of the autonomous vehicle planning system, differing response time requirements and differing needs for programing flexibility form the basis for the segmentation of major modules. The layer of the hierarchy which interacts most directly with vehicle actuators will contain elements which are designed primarily for reflexive performance and will not be required to employ any complex inference mechanisms. Each layer above this in the hierarchy will be less specialized, making greater use of knowledge-based inference mechanisms so as to allow greater flexibility and provide extended capabilities for high level reasoning.

2.7.3 Functions of Modules

As illustrated in Figure 3, the planning hierarchy is composed of mission planning, route planning, local planning, and reflexive planning modules, each receiving input from a comparably layered perception system. The mission planning module has the function of translating abstract mission goals into a set of geographic goals and constraints appropriate for use in route planning. This function requires reasoning with abstracted map features and well-assimilated sensory data, and may be allowed a response time on the order of several minutes. At the next level, the route planning module must translate geographic goals and constraints into specific route plans . This level employs computationally intensive map-based reasoning with

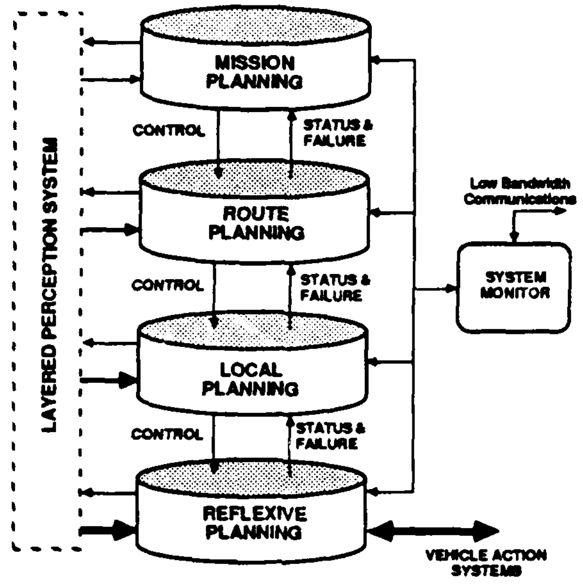


Figure 3. Multi-level Planning System

the additional use of long-term assimilated data from perception. The local planning module has the role of ensuring that a route plan gets executed properly. This module must apply heuristic planning knowledge in the context of local world models constructed from assimilated perceptual data to select reflexive actions that are appropriate to the execution of the desired route plan. This module may be limited to a response time of several seconds. Finally, at the bottom of the hierarchy, the reflexive planning module has the task of maintaining real-time vehicle control. Rapid response times are required of this module.

2.7.4 Planning Process Types

Because the planning problems at the various levels in the hierarchy are vastly different, it is clear that the type of processes which are used to implement them may be considerably different as well. In the course of developing each of the planning modules, we have encountered three distinct planning process types. These are: deterministic, stochastic, and servo control. Each of these process types has specific advantages and disadvantages with respect to its application in different problem domains. The appropriate roles for each type are highly dependent on the nature of the data available for the associated planning task and the timeliness required of the planning results.

The three planning processes may be defined as follows:

Deterministic. A deterministic planning process is one in which the available information is static and is assumed to be complete. With complete data, a deterministic planner can generate a plan which extends entirely from the starting state to the goal state. The act of selecting a route on a map is an example of a deterministic planning process since the most currently available map must be assumed complete, and the entire route from start to goal can generally be specified. An important aspect of the deterministic planning process is that it does not require that more information be obtained as the plan is executed. The static nature of the data implies that plan execution will not add information which could significantly alter the original plan, although the plan may include predetermined alternative actions to deal with known uncertainties in the data. Should information be obtained during plan execution which does invalidate the plan, the static planning process must update its data base with the new information and replan.

Stochastic. A stochastic planning process is one in which the available information is incomplete, but may be augmented as a result of plan execution. As shown in Figure 4, the state space for stochastic planning is characterized by a region of known achievable states surrounding the current state "S," and estimated states everywhere outside of that region. Since the goal state "G" lies outside of the region of known achievable states, the stochastic planning process must first select an intermediate state "B," and then generate a plan to reach that state with the objective of obtaining more information. It is important that state "B" lies within the same contiguous region of known achievable states as "S" in order for a proper plan to be generated.

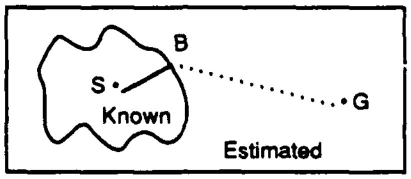


Figure 4. Stochastic planning state space

An example of stochastic planning can be seen in typical travel situations. Often, a person traveling to an unfamiliar city may not have access to maps of that city before arriving. Having general highway maps, however, one might easily plan a route to the desired city, planning to find a gas station at the new city where a map of the local area may be purchased. Thus with only partial information, a successful plan is generated based on knowledge of one's ability to acquire new information during plan execution.

Servo Control A servo control planning process is characterized by its inherent use of feedback as a mechanism for control. Servo control operates on the basis of using the difference between a current state and a goal state in order to produce a control output. The resulting control output must alter controllable state variables either directly or indirectly by cascading through other servo control processes. The changes in these variables produces a new current state which is iteratively fed back into the servo control process to yield revised control output. The cycle continues indefinitely until the goal is achieved or the process is halted through some external mechanism. Generally, when a goal state is expressed in terms of a measurable state variable, a servo control process may be used to achieve that goal.

A simple example of a servo control process is one which can guide a vehicle to a specified location by controlling its speed and turn rate. This servo process might use the vector difference between the measurable current location and the goal location as an error signal to alter speed and turn rate. Although speed and turn rate are not themselves directly controllable state variables, they are easily controllable through simple servo processes that control acceleration via motor torques. By properly controlling speed and turn rate to slow the vehicle as it approaches the goal location and turn the vehicle to aim towards the goal, the goal can be achieved.

When we consider mapping specific planning problems into the three planning process types, we find that immediacy/assimilation tradeoffs are a central factor in making these decisions. Although all three process types can be made to operate with either immediate or assimilated data, there are definite advantages to processing highly assimilated data with a deterministic planning process while processing immediate data with a servo control process. Similarly, a stochastic planning process should be applied to data which lies somewhere between these two extremes of immediacy and assimilation. The key factor in forming these distinctions is the fact that as we move from the deterministic level to the servo control level, we find that the emphasis shifts from having a great deal of static data at some instant in time to having a much smaller amount of changing data flowing in a continuous stream.

2.7.5 Intermodule Communication

With assimilation being provided by the upper levels of the hierarchy and immediacy provided by the lower levels, it is necessary for information and control to pass between levels in order for assimilated results to influence the performance of the reflexive layers. In our hierarchy, constraints and specialization commands are passed downward, and failure and other status reports are passed upward, while perception and vehicle control data flow horizontally through each level of the hierarchy. To minimize the delay through the system, only specialized reflexive procedures are allowed to issue direct vehicle action commands. These procedures, however, are activated and controlled by higher level modules performing more thorough assimilation of the environment. With a suitably large library of specialized reflexive methods and a significant knowledge base of the limitations and constraints of these methods, higher level reasoning modules are allowed to make complex navigational decisions with little regard to requirements for immediacy.

2.7.6 Module Decomposition

Although a single layer of the vertically decomposed architecture may itself be decomposed horizontally, it is possible to establish an alternative form of decomposition which incorporates a higher degree of parallelism. We refer to this as lateral decomposition. The lateral decomposition of individual modules within a vertical hierarchy is performed through the division of each module into a collection of expert agents, each capable of receiving some combination of processed sensor data and communicating with others within the same module over a common blackboard. The output of each module is then formulated by an arbiter which monitors the blackboard and is responsible for arriving at a consensus from the observed activity. This decomposition allows a very fine grained approach to module construction, where small or large enhancements to system performance can be effected through the addition of new experts to various modules without disrupting existing performance capabilities.

In addition to allowing easy and flexible system enhancement, a lateral decomposition of individual layers provides for the variety of task capabilities needed for operation under diverse and potentially hazardous conditions. As new capabilities are developed, they may be added incrementally to the rest of the system as distinct expert sub-modules, with minimal impact on existing performance standards. As the number of sub-modules grows, simultaneous activation of all sub-modules may not be meaningful. To organize the sub-modules into meaningful groups, activation sets are established. An activation set is a pre-defined group of agents which, when operating together, can produce meaningful actions. In the context of the vertical hierarchy, control from upper layers may be exerted upon lower layers through the selection of an appropriate activation set for the current situation. With this grouping of expert sub-modules into activation sets, different modes of action for the planning system can be established at each level of the hierarchy. These modes can provide different operational strategies as required by environmental changes or may be used to reduce predictability in the presence of a threat.

3 MAP-BASED ROUTE PLANNING

The Hughes AI Center has several years of experience in map-based route planning. The next section describes some previously developed work and how this work has been extended and used in the ALV project. A detailed exposition of the Hughes planning technology is in [Mitchell87].

The route planning module is responsible for translating the path constraints and goal locations provided by the mission planner into a specific route through the terrain. The route produced by the route planner is in a symbolic form which allows robust execution of the plan by the lower level modules. The primary information used by the route planning module is the digital terrain map which is static and complete to a specific resolution (typically 100 meters). Secondary information is provided by the perception system in the form of infrequent updates to the map. Because of the nature of the data, route planning is best suited as a deterministic planning process. As a deterministic process, new data arrives infrequently and unpredictably, and therefore must be accommodated with replanning methods.

To support the requirements of the route planning module, we have developed a number of route planning algorithms. Using three different representations of the digital data, three different algorithms have been implemented: 1) a grid-based planner, 2) a road network planner, and 3) a weighted region planner. The grid-based path planning algorithm represents the terrain as a tesselated grid, the road network planning algorithm uses a graph network representation of roads, and the weighted region planning algorithm represents the terrain with polygonal regions. Each of these algorithms has particular advantages and disadvantages, and is selected by the mission planner to operate on different planning problems according to its specific capabilities.

The selected route-planning algorithm produces a sequence of locations representing the planned route. The route planning module then processes the highly specific location-to-location representation of the plan into a symbolic path description for the lower level modules to execute. The symbolic path description contains key descriptions for following the route, including possible contingencies.

Since the route planning module is a deterministic planning process, the assumption is that the plan will succeed. Only unanticipated situations that are not covered by the symbolic path description and cannot be handled by the lower-level planners will make the route planning module perform replanning. We have developed a general technique for replanning which can be implemented within any of the planning algorithms.

3.1 TECHNICAL BACKGROUND

The ITAC system is an autonomous vehicle path planning system that was developed under a previous Army contract, and includes tools developed under IR&D. Since we have applied much of the planning technology developed in this system to ALV problems, a brief description of the relevant parts of this system is appropriate. We will first describe the basic planning algorithms, then explain how cost factors influence these algorithms. Finally, we describe the concept of symbolic path descriptions, which serve as the medium for communicating map plans to the sensor-based planning module.

3.1.1 Route Planners

The route planner can plan routes based on several criteria. The ITAC system has been used to plan several sophisticated routes using a DMA database of West Point (data was provided by ETL). Some of its capabilities include resource allocation, threat avoidance, and planning for specialized mission scenarios. Most of this sophistication has not been used in the ALV program as of yet.

The basic route planner of the ITAC system employs a grid-based A* search. That is, the terrain is represented as a grid, where at each grid point, there are associated costs based on the features of the terrain at that point. To find a route, the grid is searched using the A* algorithm, where the grid is represented as a network of nodes, each grid point is a node having eight neighbors. The START and GOAL may be located at any two grid points. The Euclidean distance is used as the heuristic cost estimate. The A* search would be straight forward except there is a problem with computing costs correctly using a grid representation Paths using a naive computation of cost exhibit paths that are not optimal. In fact, in a uniform cost region, a naive route planner will not plan a path in a straight line. Other researchers have implemented route planning systems but their routes have exhibited this problem. This problem and its solution are explained in detail in [Mitchell84].

In addition to the grid-based path planner, there is a road network route planner. This route planner can plan long distance routes on roads in less time than the grid-based planner. The planning time is relatively quick because it uses a two-level search. The first level search is a bi-directional search from both the START and GOAL to roads using the grid search. Once the start and goal are linked to the roads, then an A* search is performed on the road network.

As part of our IR&D effort we are developing a weighted region route planner based on the work of [Mitchel86]. This is a more symbolic planner which uses a polygonal representation for uniform terrain regions. Operating on the same principles which dictate that light will always take the shortest path through objects with different indices of refraction, this planner focuses entirely on the problem of finding appropriate entry and exit points through uniform polygonal regions. The symbolic route planner has the potential to be faster and more flexible in planning strategic routes than either of the other two planning algorithms.

The ITAC system has the ability to plan routes and provide a symbolic description of paths. Included in that description is the type of terrain the vehicle is traversing, the direction to the next goal, the distance to that goal, and the actions necessary in traversing each route segment. For example, when coming to an intersection, a path description might describe type of intersection, the way to turn, and observable landmarks in the area.

3.1.2 Stategic Costs

The grid-based path planning (GBPP) algorithm may be used to plan good routes for the ALV in a variety of different types of terrain. The ITAC system has a facility to change the cost criteria used in the evaluation of routes. This capability provides the ability to do rapid prototyping of cost functions. Included in this capability is the use of threats as part of the cost criteria.

Because of the GBPP's versatility in terms of cost structure, we can in fact impose artifici n
the terrain which may correspond to a priori knowledge we have about the costs of movement thr ,e
terrain. If there is some area which we wish to avoid, we can specify a cost mask (an array) which be
added to the objective function evaluations of grid points in that area. The GBPP will then choose
avoiding the area (provided the cost mask has sufficiently high values) or possibly going through some small part of it (if the cost mask does not have infinite values and there is no cheap way around the area).

A threat is defined to be a location on a map from which an enemy force has some means of visibility or attack. Existence of a threat usually implies that a certain region should be avoided by the autonomous vehicle in order to insure its safety. Examples of threats include enemy tanks, radar positions, and military camps. The ITAC system provides the capability to define any type of threat.

The system implements threats as instances of a "threat flavor". Included in the flavor specification are slots recording the threat's type (e.g., "tank"), location (x, y, z), visibility range (the closest our vehicle may come to the threat without the possibility of being seen), firing range (if the threat has the ability to attack), a relative weight (to express the relative importance of our avoiding this particular threat versus any other threats of which we have knowledge), and a threat risk function (to be described below).

The GBPP uses information about threats when planning optimal paths. The optimality criteria used to choose among paths is determined by the form of the objective function. An imposed threat influences the objective function via its threat risk function, whose value is added into the objective function at the time of imposition. The threat risk function, (r), indicates the risk per unit time of being visible at a distance r from a threat. One may interpret $\phi(r)$ as a probability of detection, an expected loss due to enemy fire, or any other cost associated with being detected by the enemy. The larger the value of $\phi(r)$, the greater the risk of being at a distance r from the threat. Typically, $\phi(r)$ is non-negative and monotonically decreasing in r, with a value of zero for r greater than or equal to the visibility range, R. An example of one choice of $\phi(r)$ is shown in Figure 5.

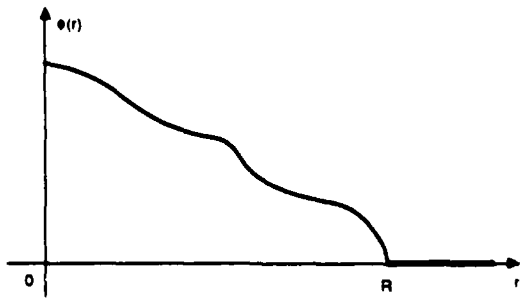


Figure 5. Example of a threat risk function.

3.1.3 Symbolic Path Descriptions

The map-based planning processes require a language for directing the lower-levels of planning. The ITAC route planner can easily generate a very specific route plan. Since the grid representation of the map is in Cartesian coordinates, a plan can consist of individual points along the path in this grid, converted into longitude and latitude, as *intermediate goal points*. This representation of a point-to-point plan is compatible with advanced navigation aids such as LORAN and GPS, which can give the position of a vehicle in longitude and latitude. In addition, the point-to-point plan is also compatible with inertial navigation systems. However, this type of description does not describe the features that will be encountered while traversing a route.

The form of description of planned routes is crucial. People can follow high-level descriptions but they can be easily confused by ambiguous or unclear directions. The key to a good description is providing the relevant features of the route that need to be recognized. Relevant and identifiable landmarks must be indicated for areas where important decisions are to be made.

If the original map data is inaccurate, then the positional data that represents some feature, such as a road, will be misleading. If the vehicle follows the positional information rather than the actual road, then the vehicle can quickly get into trouble, like getting stuck in the ditch along the road. The ideal system would be able to use either absolute positioning or relative positioning. With absolute positioning methods, a combination of global positioning and inertial navigation is used to follow a route specified by longitude and latitude points. With relative positioning, a combination of inertial navigation and sensed data are used to follow a route based on recognized landmarks and terrain features.

There is another problem with a point-to-point plan. It is the inability to specify the conditions of failure. For instance, in the execution of a plan, the vehicle might have to take a detour because of some unknown obstacles in the original planned route. This might make an intermediate point chosen by the plan unreachable. In some cases, the intermediate point is not critical, but in other cases the failure to reach the point can jeopardize the mission. Moreover, the amount of deviation from the planned route is more critical at some times than in others. If a detour bypasses, say, a point that satisfies a mission goal, then the plan has failed. On the other hand, a detour may be perfectly acceptable if it eliminates points on a path whose only function is to get to the goal

To alleviate the problems encountered when using a specific route description, a more general route description is useful. The generalized route description has the advantage of eliminating the overly specific intermediate goals. However, the generalized route puts the burden on the local path planner to generate local goal points and recognize the objects that the path description uses to specify the route. For example, the generalized description "Head south until encountering a road" implies that the local planner can recognize a road. An alternative approach is to combine both the specific point-to-point plan and the generalized path description. They can be combined by forming a hierarchy of descriptions, where the point-to-point plan is the lowest level and a generalized route plan is at the higher level, as in Figure 6. This type of representation gives the vehicle the capability of using the highest level of description that it can recognize in the current situation. For instance, if it is having difficulty with recognizing a road, then it falls back to using the more specific point-to-point locations that, according to the map, are locations on the road. If at a later date the road is reacquired by the perception system, then the higher level description will be used.

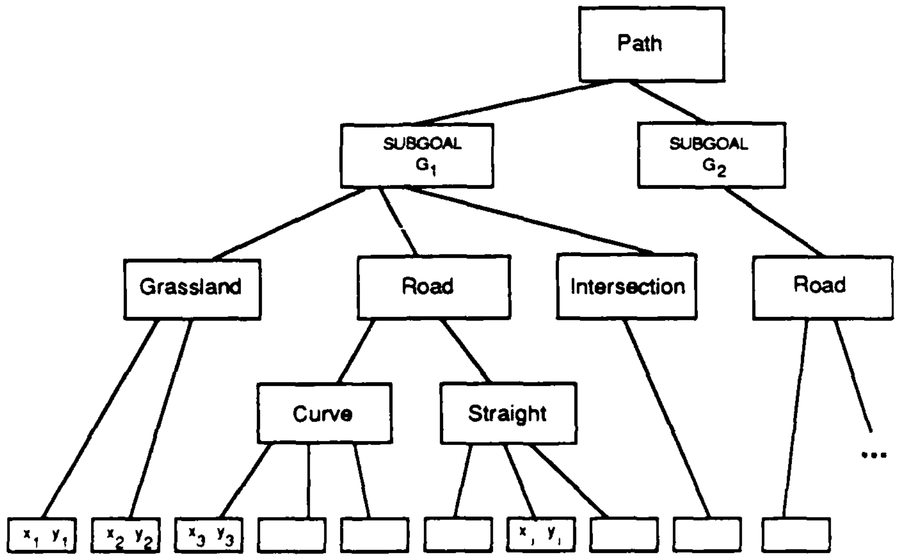


Figure 6. Hierarchical Symbolic Path Description

The symbolic information for path description may be very diverse. The important parts of the symbolic information are the recognizable objects and attributes which have an impact on the decisions that must be made by the autonomous vehicle.

Symbolic path descriptions have two important components: decision points and verification intervals. A decision point is a place along the route where a choice must be made. Descriptions of decision points may include landmark and terrain features (for confirming that the vehicle has arrived at the decision point) as well as action and heading information to insure that the proper decision is made. For instance, the decision point D3, shown in Figure 7 is a point in which the vehicle must choose between two roads. In this case, the action is to chose the left road. Included in the decision point is all the information needed to make that choice, such as knowledge that the lake is off to the right.

A verification interval is a section along the route that can be used to confirm or deny that the vehicle is following the desired path. In addition, it specifies constraints for dealing with unexpected situations along the path. Descriptions of verification intervals include landmark and terrain features and they define areas along the path that these features should be encountered. For example, consider again Figure 7. The verification interval V6 indicates that the lake is on the right, while the interval V7 indicates that the lake is on the left. If the vehicle discovers that the lake does appear on the left then interval V7 would have

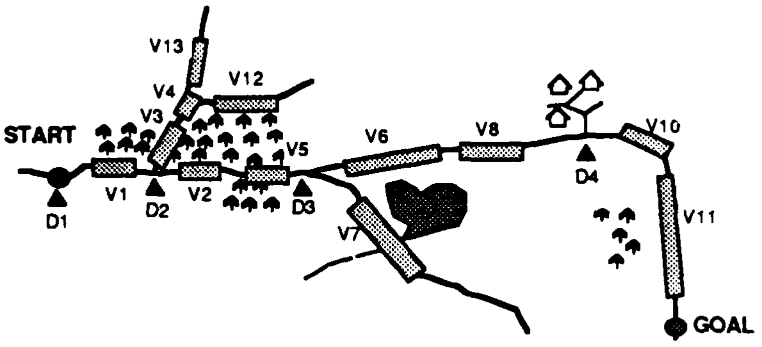


Figure 7. Terrain Map with Symbolic Descriptions

information on what was the mistake at the previous decision point, D3, and how to correct it. Verification intervals may also have constraints that limit the range of decisions that can be made in unexpected situations in order to satisfy mission goals. For example, the vehicle may be constrained to avoid marshes. In the event that an obstacle must be avoided when traveling near a marsh, the vehicle will have a strong preference to head in a direction that leads away from the marsh.

As shown in Figure 8, a symbolic path description represents both the correct route and a collection of possible incorrect branches from the desired route which could be reached if an improper choice is made at any decision point. When the vehicle is traversing the path correctly, verification intervals will serve to build confidence in the correctness of past decisions. In contrast, recognized landmarks or terrain features may correspond to descriptions in verification intervals that branch off of the desired path. This correspondence will contribute doubt to the choices made at previous decision points and may finally result in a decision to go back to the most likely place where the failure could have occurred.

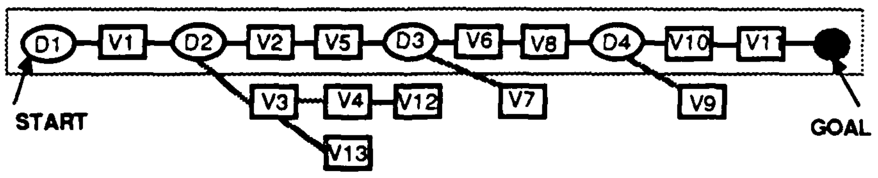


Figure 8. Symbolic Path Description

The landmarks and terrain features of symbolic path descriptions are recognizable objects expected in the immediate environment of the vehicle. Any feature that can be readily identified as being unique will play an important role in navigation. Since man-made objects are the most recognizable objects for machine vision, they will play a significant role in symbolic path descriptions. Roads, intersections, buildings, tunnels, and bridges are examples of recognizable man-made objects. They also appear in digital map data, and therefore they can be used extensively in symbolic path descriptions.

Most natural features are not distinct enough to provide accurate navigational information. However, a natural feature will have a high value for navigation if it occurs infrequently in the terrain and has distinct attributes. Small lakes and ponds, for example, are rare in most terrain and have distinct boundaries, giving them a high navigational value. Rivers provide good boundary information, but they rarely provide good reference points for accurate positioning. Hills do not usually provide valuable reference points because of the ambiguity of their boundaries and shape. Valleys and mountains are useful reference points at a distance.

but not at close range. General terrain type features such as forest and grassland can help to verify the correctness of a path, but cannot be used for reliable positioning.

3.1.3.1 Building Symbolic Path Descriptions

There are two sources of information to be used in building symbolic path descriptions. The first source is the route planner. The route planner provides information on the important decisions it made in planning the route. This would include consideration of preferred terrain for travel, path bottlenecks, tactical threats, and resource conservation. With an explicit representation of this information, the symbolic path description can specify route plans that satisfy plan goals without necessarily following the exact point-to-point route.

The second source of information is provided by a reasoning system that analyzes the path from the route planner in the context of the map and generates appropriate decision points and verification intervals. This reasoning system structures map information in a form that is compatible with the perception, reasoning, and decision capabilities of the vehicle as it traverses the route. In this way, navigational cues can be expressed in terms of landmarks and terrain features along the route that should be recognizable by the vehicle. In addition, this reasoning system can determine the path constraints that are used within verification intervals to help deal with unexpected situations.

3.2 DEVELOPMENT METHODOLOGY

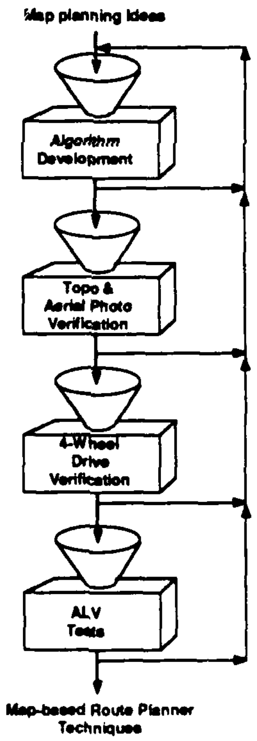

Map planning Ideas
Algorithm Development
Topo & Aerial Photo Verification
4-Wheel Drive Verification
ALV Tests
Map-based Route Planner Techniques

Figure 9. Map-based Route Planner Development Cycle

We view the development of map-based plans for the ALV as an iterative process. Given that the amount of time to test routes on the ALV is limited, and the operational difficulties in using the ALV prevent extensive testing, it is important to have well tested route planning techniques before using them on the ALV.

The development of a map-based planner and its heuristic cost functions is viewed as a four stage process (see Figure 9). The first stage is the initial idea and an implementation of it in the map-based planner software. The second stage is testing the implementation by producing planned routes and verifying the reasonableness of these plans by looking at both the topographic map and aerial photos of the area. Most of the fine tuning of the techniques should be done at this stage. The third stage is to verify some of the planned routes by actually traversing them at the test site with a 4-wheel drive vehicle or by walking the intended routes by foot. Data gathered in this way can be used to change or refine the route planner and its heuristics. Once the route planner passes these verification stages, then the last stage, traversing the planned routes with the ALV, must be performed. Once the ALV traverses these routes, the data is analyzed and either the route planning techniques are refined or different techniques must be devised and the whole process is started over again.

3.3 PROGRESS

This year's progress in map-based planning technology included database conversion, development of methods for updating a land navigation system, and development of a new planning technique which copes with dynamic conditions. The database conversion has involved the adaptation of the ETL-Martin Marietta map database into a format which is suitable for use in the ITAC planning system. The land navigation update techniques, as part of a symbolic path description, exploit a database of landmarks and geometric reasoning techniques to refine estimates of the vehicle location. Finally, for dealing with mobility costs which may vary with time, we have developed a planning technique which provides optimal routes under these circumstances.

3.3.1 ETL-Martin Marietta Database Conversion

To use the ITAC map-based planner, the Martin digital terrain database had to be converted. The Martin terrain database, provided by ETL, included: 1) Landcover information, 2) Roads, 3) Gullies and Ravines, and 4) Elevation.

The polygonal representation of the landcover has been converted intc a grid representation. This was accomplished by displaying the polygons in a raster-scan display buffer at the required resolution and saving the result. The road data was converted into a spatially indexed network representation. This conversion was based fairly directly on the arc-node representation of the raw data. The gully information was also eventually converted into network representation, but in the beginning it was converted into a grid representation.

The elevation data was provided in stages by ETL. A low resolution version of 30 meter resolution was provided first. We interpolated this to a 5 meter resolution and incorporated it into the ITAC system. Later, a preliminary high resolution version of the elevation data was provided, and we merged it into the interpolated version. This was done because the high resolution version covered only a small patch of the test area.

Figure 10 illustrates a typical route plan generated by the ITAC system. The planning criteria for this route were to ignore roads and avoid change in elevation. Most terrain along the route chosen by the planner was relatively flat. The planner chose the most advantageous point along the gully to .oss, having no alternative means to get to the goal.

3.3.2 Updating the Land Navigation System

To assure more robust execution of planned routes, we have developed methods for reasoning about landmarks and their relative geometry to refine vehicle position estimates. There are several methods for

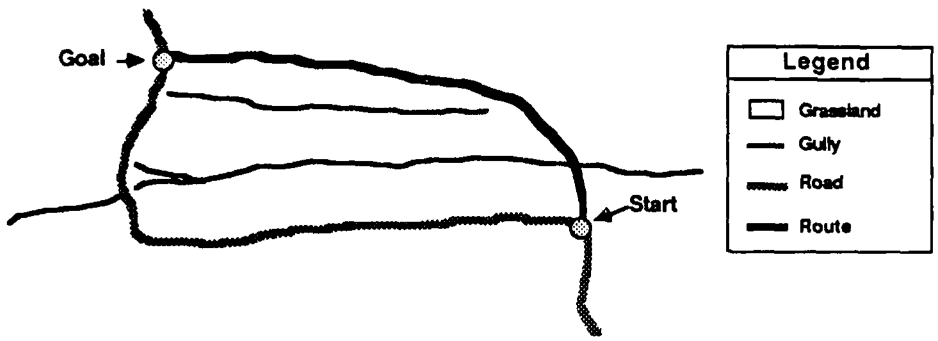


Figure 10. Map-based Route Plan

using landmarks and other terrain features for position update. We have characterized these methods and can apply this knowledge to predict suitable areas along a path for performing such updates.

There are two broad classes of position update situations that arise in the ALV application: updating while on a road, and updating while off road. The updating of the vehicle location when it is on a road has five basic configurations that can arise: 1) a landmark which is close to the road or part of the road (i.e. an intersection) is within a known range to the vehicle, 2) the direction to a landmark is known, 3) two colinear landmarks are visible, 4) the direction to two landmarks is known, 5) the direction to three landmarks is known. The updating of the vehicle location when it is off road has three basic configurations: 1) the vehicle is within a known range of a landmark, 2) the direction to two landmarks is known, 3) the direction to three landmarks is known. Each configuration has different applicability and provides a different amount of error reduction. In general,the more applicable configurations tend not to be as good at reducing error as the less applicable configurations. One function of the planning module is to provide descriptions at various points along its planned paths which assist in the selection of appropriate error reduction techniques.

The Figure 11 represents a configuration for vehicle position update based on the vehicle being on a road and detecting two colinear landmarks.

Let, Σ_{L_1} and Σ_{L_2} be the maximum uncertainty in position of the landmarks L_1 and L_2 respectively. Let D_v be half the distance between landmarks plus the distance from L_2 to the vehicle on the road. Let the angle θ represent the angle of the road relative to the line of sight to the landmarks. Assuming the road is straight and that both Σ_{L_1} and Σ_{L_2} are very small relative to the distance, $DIST$, between landmarks, then the error along the road, D_ϵ, would be

$$D_\epsilon = \frac{D_v \sin(\arctan \frac{2(\Sigma_{L_1}+\Sigma_{L_2})}{DIST})}{\sin\theta}$$

Each type of landmark configuration can possibly reduce the amount of uncertainty in the LNS depending on the errors involved. At each point along a path, a landmark configuration can be evaluated to see how much of an error reduction can be accomplished. If there is sufficient potential for error reduction, and the current plan warrants the expenditure of computational resources (e.g., the resources to recognize the landmarks), then the landmark update is performed. We are investigating how to represent these configurations in the symbolic path description of the ALV map-based planner.

3.3.3 Planning Strategic Paths Under Dynamic Conditions

Current route planners do not adequately take advantage of knowledge of expected changes in the environment. Their only response to possible changes in the environment is to replan when altered conditions

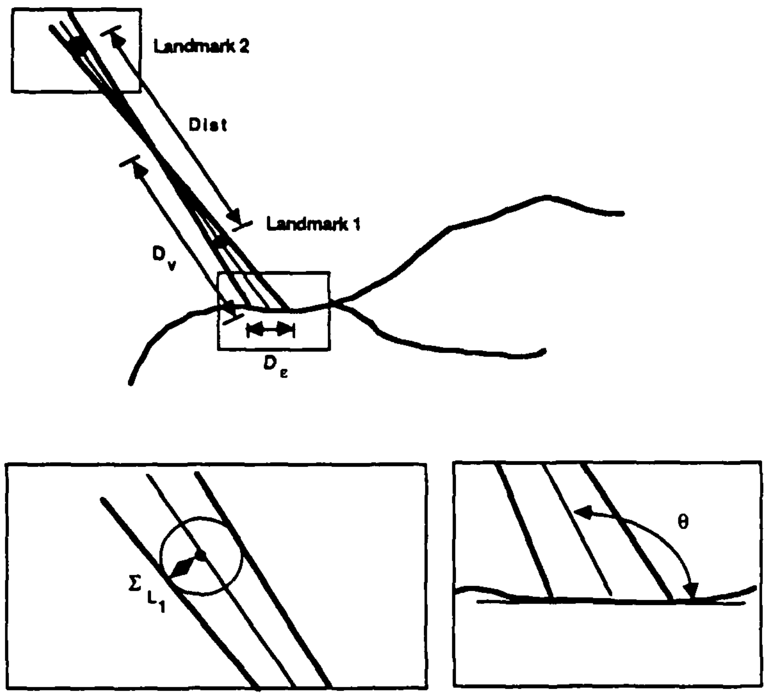


Figure 11. Road Positional Update by Colinear Landmarks

arise during plan execution. We have been exploring methods for planning paths considering both dynamic and uncertain conditions. As an initial step we have developed a method of finding optimal paths through terrain where the cost of traversal is solely a function of time. This makes map planning applicable to conditions where bridges are expected to be useful only for a limited time period, or where traffic conditions on roads may vary with the time of day, significantly affecting one's travel time.

Here we will explore possibilities for an intelligent planning of paths through weighted digraphs (usually representing a digital terrain map or another abstraction of an environment) in a situation that is dynamic or uncertain, or both. This means the weights on the arcs need not to be constants, but rather they may be functions, random variables, or parametric random variables.

It will be more convenient to talk only about graphs and digraphs with no multiple edges and arcs. Therefore we shall assume that for any problem enough nodes are inserted so the graph becomes simple. Conceptually nothing is changed.

We shall always assume that the arc parameters represent the **time** required to travel the arc in the given direction, e.g. $c_{ij} \geq 0$ is the time to reach node j from node i when traversing the arc (i, j). The parameters will be referred to as the **weights, costs,** or **transition times.**

For a given digraph $D(V, A)$ and a given pair of nodes $a, z \in V(D)$ the underlying objective (loosely speaking) of all the problems to be considered here is to find a walk (i.e. nodes and arcs may be revisited) that starts at a node a, terminates at a node z, and "walks" from a to z "fast."

First we make a short recapitulation of the static deterministic case. It is well studied in the literature under the name **Shortest Path Problem** (usually abreviated **SP**), where it is shown how the problem can be formulated in the contexts of *Linear Programming*, and *Dynamic Programming*. An efficient algorithm for solving the problem is the *Dijkstra Algorithm* which can be derived from the *Primal-Dual Algorithm* for linear programming problems, or directly from the *Bellman Equations* in the dynamic programming setting. The asymptotic complexity of this algorithm is $O(|V|^2)$, where $|V|$ denotes the cardinality of the set of vertices of D. Heuristic algorithms A^*, and A^{**} are extensions of the Dijkstra Algorithm which are of the same **asymptotic** complexity, but which tend to improve the **average** running time on a large and important class of problems.

As for the practical aspects of implementation we suspect that at present, the Linear Programming interpretation of the problem has not been exploited to the fullest. When the min-path computation has to be done from "scratch," the Dynamic Programming approach is probably going to be the approach of choice. However, if one is to admit sudden mishaps, and therefore the necessity to reconsider the path used in the middle of an actual walk, then the possibility of a *"hot start,"* and the availability of *primal-dual pivots* as well as availability of some new and fast algorithms based on *Monotropic Optimization* (or even *Dual-Interior Methods*) could turn out to be valuable.

Inherently, there is to be a little circularity in the dynamic class of problems considered by us, since the natural parameter of the functions $c_{ij}(.)$ is in our case T, i.e. the time which we are trying to minimize. This makes the dynamic case intriguing as well as (curiously enough) tractable. Here we can argue that through some reductions even a very general formulation of the dynamic generalization of the min-path problem is essentially solvable by the Dijkstra Algorithm, A^* algorithm, or A^{**} algorithm. These algorithms must all be started in the "source" vertex a, which is a significant difference from their respective static counterparts, where the algorithms can be equally well rooted in the source a as in the sink z. Another reason why the dynamic problem is harder than the static version is that the Linear Programming formulation is no longer available and therefore efficient updates may be harder to obtain. Nevertheless, we still can solve these problems in $O(|V|^2)$ number comparisons and $O(|V|)$ updates (as it is in the regular Dijkstra Algorithm), and $O|A|$ function evaluations. Whether the function evaluation is simply a look-up procedure or is (in a worse case) a one-dimensional search depends on the the nature of the functions and on the level of preprocessing.

Assume F is the class of functions (say $f \in F$), which satisfy these mild conditions:

i) $f(x) \geq 0$ is a piecewise continuous function,

ii) $f(x)$ is either real, or plus infinity

iii) $f(x)$ is lower semicontinuous, i.e. $f(x) \leq \liminf_{y \to x} f(y)$

iv) $f(x)$ has only finitely many discontinuities over compact intervals.

If each of the transition time functions $c_{ij}(T)$ belong to this class F then an optimal solution can be found by one of the above mentioned algorithms. The class F is not very restrictive in the sense that all practical problems we treat give rise to the functions within this class.

The class of functions F poses several important invariance properties that can be utilized theoretically as well as practically. In particular, extensive use has been made of the closedness under the taking of pointwise minima. This allows us to build the functions from the "primitives". For instance we can combine different routings (i.e. multiple arcs) between two locations into one, and we can combine different means of transportation between the locations into one resulting function. Other important properties of F are that the class forms a nonnegative convex cone (i.e. F is closed under pointwise addition, and under multiplication by the nonnegative scalars), and that the class is closed under translation. The property iv) prohibits this collection of functions from being closed topologically.

The class of functions F contains an important subclass, namely that of all **piecewise linear functions** (p^1 functions) satisfying properties i)-iv). Lets call this subclass P. and note that this subclass inherits all the important properties from F. It also inherits the properties as the entire class, not just as the individual functions. Class P forms a nonnegative convex cone, closed under the translation and under the extraction of pointwise minima. Again P is not closed topologically, and therefore the only real distinction from the class F is the fact that P is not closed with respect to the pointwise multiplication.

In more precise notation the closure properties mean that for any f and $g \in F$ (or P) the following are true:

i) $(f(x) + g(x)) \in F$ (or P),

ii) $\lambda f(x) \in F$ (or P) for all $\lambda \geq 0$,

iii) $f(x+t) \in F$ (or P) for any fixed real parameter t,

iv) $(f \vee g)(x) = min\{f(x), g(x)\} \in F$ (or P),

and for the completeness we also list the multiplicative property:

v) $f(x) \cdot g(x) \in F$.

A crucial property for us is the fact that P is dense in F. By this we mean that if we impose the convention that the difference of two positive infinite numbers is 0, and the assumption that $\mu(x) \geq 0$ is a finite measure, then for any $f \in F$ and $\epsilon > 0$ there exists for instance a function $g \in P$ such that

$$\int_{-\infty}^{+\infty} (f(x) - g(x))^2 \, d\mu(x) \leq \epsilon.$$

This is to say that any function of the class F can be approximated arbitrarily close (measured here by the L_2 norm) by a function from the subclass P. Piecewise linear functions can be represented by efficient data structures. Also a large number of functions from practical applications are piecewise linear. It is therefore only natural to choose the p-l maps (with a finite number of linear segments) as the input functions for practical implementations.

We will agree that the size of the input for a piecewise linear map can be measured by the number of the linear segments. Defining the input size this way, we can show that to do a one dimensional line search (to find a minimum) requires a linear time, while the look-up procedure for the evaluation of the p-l map is logarithmic in the input. If we preprocess the p-l function (in order to be able to do the function evaluations instead of the line searches) we will obtain another p-l map of the same input size. The preprocessing is being done in the linear time. In the same way, only a linear time is required to create a p-l map that is a pointwise minimum of two p-l functions.

By associating the traversal time cost along various arcs of a graph with functions of class F, or their approximations from class P, it is possible to apply the Dijkstra, A^*, or A^{**} algorithms to determine an optimal path under dynamic conditions. As the search proceeds from the start toward the goal in the Dijkstra algorithm, a wavefront is propagated whose perimeter represents all points with equal cost from the starting point. When the wavefront finally reaches the goal, the steepest descent along these perimeters yields the lowest cost path. With path cost now being equated with traversal time, and with traversal time from a node being a function of the arrival time at that node, the wavefront propagation is fundamentally the same. The only difference is that the time at which the wave reaches a node may have a direct influence on the time required to leave that node.

4 SENSOR-BASED PLANNING

4.1 BACKGROUND

In the context of our hierarchical planning architecture (see Figure 3), all sensor-based planning issues are primarily handled by the local planning and reflexive planning modules. The distinction between these two modules may be viewed in terms of their relative reaction times and the types of planning processes they each employ. The reflexive planning module performs all tasks requiring immediate response to new sensor data while the local planning module analyzes assimilated sensor data in order to appropriately alter the performance of the reflexive planning module. As a result of these performance requirements, the reflexive planner is designed as a manager of multiple servo-control processes while the local planner employs stochastic planning techniques in order to make partial plans which anticipate the acquisition of new information. The distinction between these two modules provides a clean separation between algorithmic control problems and tactical maneuvering problems which may require more symbolic reasoning methods. We will first discuss the reflexive planning module which provides all action primitives to the planning system and then discuss how the local planning module is used to control the vehicle through its influence over the reflexive planning module.

4.1.1 The Reflexive Planning Module

The reflexive planning module is critical to the achievement of real-time control because it provides the required high-throughput data paths between sensing and action, while requiring only low bandwidth interfaces to the higher levels of control. This results in the isolation of tight control loops from other reasoning tasks, adding to the flexibility of higher levels of reasoning while permitting real-time optimization at the lower levels. All high-throughput data paths flow in parallel through a large collection of expert sub-modules. Each of these sub-modules is capable of making primitive decisions about vehicle actions under specialized circumstances, using specialized perceptual data. The local planning module exerts control over the vehicle through the activation and de-activation of select groups of these sub-modules over fairly low bandwidth message-passing links. We first describe the structure of the expert sub-modules, then show how these units are combined into groups called activities. Finally, we will discuss various issues of communication and control for reflexive planning.

4.1.1.1 Virtual Sensors and Reflexive Behaviors.

Each expert sub-module of the reflexive planning module, as shown in Figure 12, is divided into two distinct elements: a perceptual component called a virtual sensor and an action component called a reflexive behavior or a behavior. This division is useful because it allows the sharing of assimilated perceptual data between a variety of different reflexive behaviors. When one of these sub-modules is operating, a portion of the available sensor data flows through the virtual sensor where it is processed and abstracted. This abstracted data is then fed into the reflexive behavior which evokes vehicle control commands based on this data. Under normal circumstances, several virtual sensors and reflexive behaviors are operating asynchronously and in parallel. Communicating through a common blackboard, their vehicle control decisions are fed through a command arbitration unit which selects the highest priority commands and issues them to the vehicle actuators.

<u>Definition of Virtual Sensors.</u> Virtual sensors may be regarded as black box sensing devices or perceptual units which can detect very specialized environmental features. Under our use of the term, virtual sensors are purely conceptual entities and need not necessarily be identified as distinct units of hardware. Internally, a virtual sensor is simply a sensing and processing unit which provides assimilated sensor data. The degree of assimilation provided may vary significantly from one type of virtual sensor to the next, depending on the type of features the sensors are intended to detect. A specific process or cascade of processes which

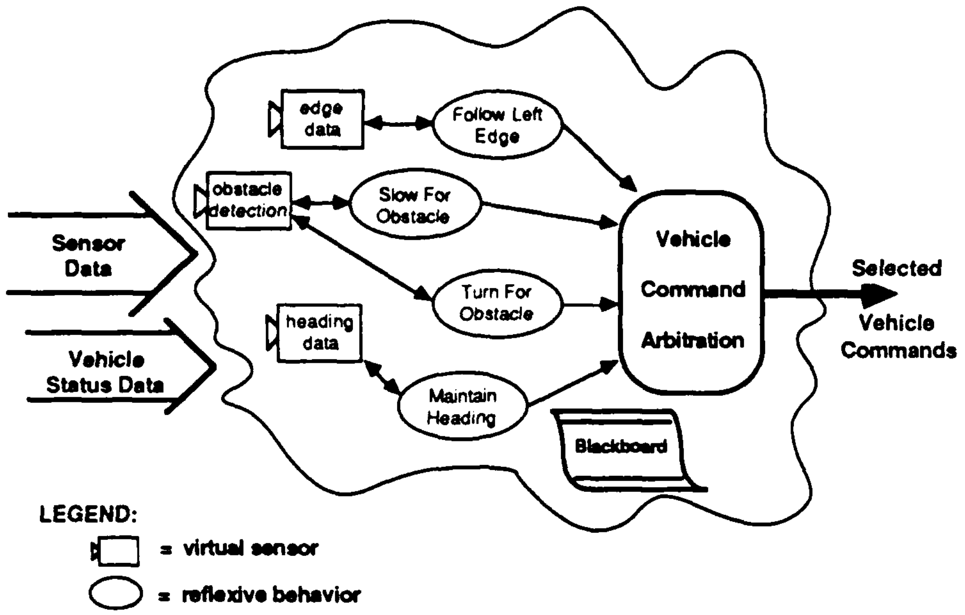


Figure 12. An activation set of reflexive behaviors and their virtual sensors

assimilate data from one or more sensors may be identified as a virtual sensor. In the context of Figure 2, virtual sensors are most appropriately viewed as part of the layered perception system. In this regard, they represent the tightest coupling between perception and planning for the whole system.

Definition of Reflexive Behaviors. Reflexive behaviors are highly procedural units due to the high demands for immediacy. The pseudo-code example below of a simplified version of the *slow-for-obstacle* behavior provides an illustration of some of the important features of a reflexive behavior definition:

```
BEHAVIOR: slow-for-obstacle
  Parameters: slow-distance = 20;
              stop-distance = 5;
              deceleration = 1
  Virtual sensors:
     (sensor-type: obstacle-detection update: 2) => distance
     (sensor-type: speedometer update: 3) => vehicle-speed
  DO Forever:
    PAUSE
    Wait For New Data IF TIMEOUT, Put Command SPEED = 0
    IF distance < slow-distance
       THEN:
         IF distance < stop-distance
            THEN: Put Command SPEED = 0
            ELSE: Put Command
                      SPEED = vehicle-speed - deceleration
```

The key elements of this definition are the parameter declarations, the virtual sensor assignments, and the procedure specification. The parameter declarations allow the assignment of a set of default parameter values which may be altered under different activation contexts. The virtual sensor assignments define the sensor inputs to the reflexive behavior. In the example above, inputs are expected to be received from two virtual sensors, one called *obstacle-detection* and another called *speedometer*. During operation, the continuous data flow from these virtual sensors will be assigned to the two local variables *distance* and *vehicle-speed*. The procedure specification for the behavior is written as an infinite loop because it is intended to operate as a continuous independent process. The process is controlled within the loop through a *PAUSE* statement and a command to wait for new data. The *PAUSE* forces the process to wait for an update period which is determined from the update rates requested for the virtual sensors. This is intended to prevent the process from unnecessarily consuming shared processing resources. The "*Wait For New Data*" command forces the process to wait for new data from the virtual sensors. This prevents the re-use of obsolete sensor data and allows the process to handle sensor failure conditions through use of the TIMEOUT condition. Within the body of the reflex procedure, the "*Put Command*" operation is used to set SPEED to various levels. This is the mechanism through which the behavior controls vehicle action. With a group of reflexive behaviors operating concurrently, prioritized commands for both SPEED and TURN-RATE are issued and those commands with the highest priority are selected for vehicle control.

Behavior Activation Sets. The most common way to provide vehicle control is through the concurrent operation of groups of reflexive behaviors, called activation sets. Activation sets are composed from subsets of the total collection of defined behaviors. Figure 13 illustrates an activation set selected from the pool of available reflexive behaviors. During operation, only the behaviors from the current activation set are enabled to process data and elicit vehicle commands. Similarly, only the virtual sensors which support these behaviors need be active as well. In general, a single activation set is fairly specialized in its suitability to different environmental situations. With a sufficiently large library of reflexive behaviors, however, a wide variety of different activation sets may be generated to provide a diverse set of task capabilities suitable to most any situation.

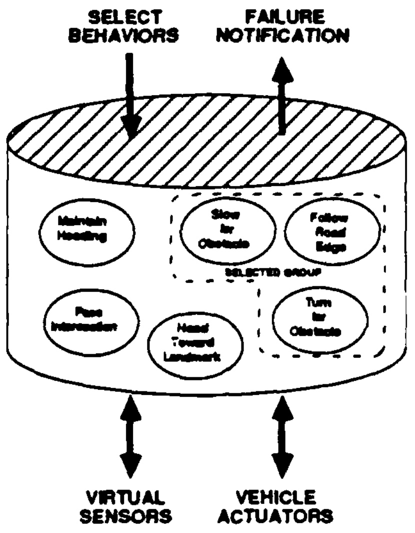


Figure 13. Activation sets are selected from a pool of reflexive behaviors.

4.1.1.2 Activities.

Because activation sets play such a key role in vehicle control, it is useful to have a standardized method for defining meaningful activation sets. The method is based upon describing the unique collection of attributes of the activation set. The method is called "activity definition," and an instance of an activation set thus defined is called an "activity." In addition to identifying the set of reflexive behaviors that must operate together, an activity definition may specify the following attributes of activities:

- **Relative behavior priorities.** It is necessary that there be a way for selecting a single command from among all the commands concurrently issued by different behaviors. With command priority as the primary criterion for this selection, each behavior places an integer priority on all of its commands. This alone, however, is not sufficient to guarantee priority uniqueness among behaviors of an activation set. Therefore, an activity definition places an ordering on its behaviors, establishing relative priorities which are used to select among commands that would otherwise have equal priority.

- **Behavior parameter constraints.** Each behavior within an activation set is a parameterized procedure. Often, when behaviors are combined in different ways, it is desirable to modify behavior parameters so that cooperating behaviors can function more compatibly. An activity definition may specify parameters for any of the component behaviors, allowing specialized tailoring for maximal compatibility. Two activities composed of the same behavior set may actually be capable of performing entirely different functions if they are given different parameter constraints.

- **Initialization and termination actions.** Both before and after an activation set is enabled, it may be desirable to invoke some specialized procedures for performing certain set up or completion operations. Activity definitions can specify any number of independent initialization and termination actions as needed.

- **Sub-activity sequences.** Often, by stringing together a sequence of activities over time, the vehicle can be made to execute simple scripts. Although normally the task of sequencing activities belongs to the local planner, certain scripts may be so common that they warrant definition as activities themselves. To allow for this, activities can define sub-activity sequences. When an activity containing a sub-activity sequence is invoked, the activity will iteratively invoke the activities within the sequence until one of them causes the iteration to halt. If an activity is listed as a sub-activity of itself, it too will be invoked in sequence along with the other sub-activities. When invoked in this way, however, the activity's behavior set will be activated rather than its sub-activities. This provides a mechanism for the activity to perform transition operations between activation of its different sub-activities.

- **Failure handlers.** Under normal operation of an activation set, there are often a number of typical failure conditions that may arise. Because the actions of most activation sets can be characterized fairly well, it is also possible to determine appropriate response activities for certain failure modes. Should a failure occur, it is desirable for control to temporarily be transferred to the failure handler for corrective action, then returned back to the initial activity after the handler is finished. An activity allows the specification of failure handlers of this type for any number of failure modes.

- **Command arbitration rules.** Although there exists a default method for command arbitration, it is possible that a particular combination of behaviors may function most effectively with some alternate approach. To maintain maximal flexibility, activity definitions support this option by allowing special arbitration rules to be in effect only while the activity is enabled.

Examples of Activity Definitions. The following examples of the *WANDER*, *SEEK-HEADING*, and *BACK-AND-TURN* activities illustrate some of the features of activity definitions:

```
ACTIVITY:  WANDER
  BEHAVIORS:      (slow-for-obstacle turn-for-obstacle)
  PARAMETERS:     ((turn-for-obstacle turn-distance 25)
                   (slow-for-obstacle slow-distance 15))
  SUB-ACTIVITIES:  Nil
  FAILURE-HANDLERS:   ((collision BACK-AND-TURN))

ACTIVITY:  SEEK-HEADING
  BEHAVIORS:      (slow-for-obstacle turn-for-obstacle maintain-heading)
  PARAMETERS:     ((turn-for-obstacle turn-distance 30)
                   (slow-for-obstacle stop-distance 10))
  SUB-ACTIVITIES:  Nil
  FAILURE-HANDLERS:   ((collision BACK-AND-TURN))

ACTIVITY:  BACK-AND-TURN
  BEHAVIORS:      Nil
  PARAMETERS:     Nil
  SUB-ACTIVITIES:   (BACKUP-BRIEFLY TURN-ON-AXIS)
  FAILURE-HANDLERS:   ((collision FORWARD-AND-TURN))
```

The *WANDER* activity is designed to cause the vehicle to wander aimlessly through the environment without hitting any obstacles. The *SEEK-HEADING* activity is similar to the *WANDER* activity, but it also attempts to orient the vehicle toward a specified heading whenever possible. It is interesting to note that the main difference between these two activities is the addition of the *maintain-heading* behavior within the *SEEK-HEADING* activity. Since the behavior ordering within the *SEEK-HEADING* activity gives *maintain-heading* the lowest priority, *SEEK-HEADING* will operate just like *WANDER* except when no obstacles are present to activate the first two behaviors.

The *BACK-AND-TURN* activity is used as a failure handler for collisions in both of the other two activities. This is used with the idea that since both activities are expected to cause the vehicle to travel forward, a logical solution to most collision situations should be to backup slightly and turn. To perform this operation, the *BACK-AND-TURN* activity must sequence through the *BACKUP-BRIEFLY* and the *TURN-ON-AXIS* activities, providing a good example of the use of sub-activities to define a simple script.

4.1.1.3 Intramodule Communication.

The critical communication pathways within the reflexive planning module are primarily between virtual sensors and behaviors, between behaviors and their local blackboard, and between behaviors and the command arbitration unit. All other communication, such as that required for activity selection or failure notification between the reflexive planning module and the local planning module, occurs at a very low bandwidth and does not raise serious performance concerns. For communication between virtual sensors and behaviors, specialized high throughput data channels must be available. When a behavior requests a virtual sensor, such a channel must be dynamically allocated to serve as a data pipe between the two processes. Fortunately, external synchronization is not required. The virtual sensor and the behavior will both run at their own rates. If the virtual sensor produces more data than the behavior can handle, then the excess data will simply be flushed from the pipe. If the virtual sensor does not provide data fast enough for the behavior, then the behavior will wait until it reaches its timeout limit.

Communication between reflexive behaviors and their local blackboard is optional. In fact, in an implementation of the reflexive planner on a highly parallel machine architecture, it is likely that communication along this pathway would be avoided altogether. When it must be used, the blackboard is sufficiently specialized to allow efficient information flow. Blackboard entries are simplified such that they may contain only a command and a value. For the sake of efficiency, the set of possible commands is pre-compiled before run time. In this way, the blackboard acts very much like a shared variable space except that the rules of command arbitration described below apply whenever an entry is made.

A special portion of the local blackboard is used for the vehicle control commands of SPEED and TURN-RATE. Any active behavior may enter a value for either of these commands, but the command arbitration rules determine whether or not the entry will be accepted. To transfer accepted commands to the vehicle control system, an independent process monitors these two special entries of the blackboard, and periodically relays any new values that it finds. The period of this process must be no less than the period of the highest priority behavior if correctly arbitrated commands are to be issued.

4.1.1.4 Command Arbitration and Selection.

The default rules for arbitrating between simultaneously received commands are quite simple. Since each command entry may include a priority, commands of higher priority replace those of lower priority. Should two commands of the same priority be issued, then the priority ordering established within the activity definition is used to determine which will prevail. To perform this selection, every command entry must include a reference to the behavior that issued it. Should the same behavior issue a new entry for the same command with the same priority, that new entry replaces the old one.

4.1.1.5 The Control Mechanism.

The reflexive planning module features an underlying mechanism that effects the change of activation states. The control mechanism performs operations necessary for efficient transitions in four types of state changes:

- **Initializing an activity.** Whenever a new activity is initialized, each of its component behaviors is initialized as an independent process, with local parameters assigned from the parameter specifications of the activity. In addition, the virtual sensors that feed the behaviors are initialized and data links are established between the behaviors and their virtual sensors.

- **Switching activities.** When a new activity is invoked to replace one that is already in effect, the behaviors of the active one are disabled and their data links are released. For efficiency, the virtual sensors and data links required for the new activation set are initialized in advance of the actual switching operation.

- **Handling failures.** Handling a failure condition through an activity's failure handler is similar to switching activities, except that control is returned to the original activity after the failure is handled. As with activity switching, best performance is obtained if all of the virtual sensors and data links for each of the error handling activities are initialized before any errors occur. Unlike in activity switching, however, the virtual sensors and behaviors of the original activity are merely put on hold while the failure is being handled so that they can readily be re-activated afterwards.

- **Executing simple scripts.** Executing a sequence of sub-activities as part of the execution of a main activity is a means for performing simple scripts. Naturally, this involves the same concerns of activity switching, but also takes into account added knowledge of future virtual sensor and data link requirements. With this knowledge, advance initialization is used to achieve greater efficiency.

4.1.2 The Local Planning Module

The local planning module has the responsibility of translating symbolic route descriptions provided by the route planning module into appropriate sequences of reflexive activities to be executed by the reflexive planning module. As shown in Figure 14, the local planning module receives its goals from the route planning module in the form of a symbolic path description; it receives knowledge of the vehicle environment from scene models provided by the perception system; and it receives operating status and failure information from the reflexive planning module. Internally, the route description and path constraints are translated into specific action goals, the scene models are used to construct a local map of the vehicle's environment, and the behavior status data are monitored to ensure that the current action goals are being satisfied. The primary local planning function of selecting appropriate reflexive activities is performed through the assimilation of action goals, local map data, and behavior status data with respect to knowledge of reflexive behavior performance characteristics.

Because of the incomplete nature of the data available to the local planning module, it is best implemented within a stochastic planning process. The symbolic path description provided by the route planner may completely specify the desired route, but in the process of executing this plan, the local planning module requires information about the local vehicle environment. This information, however, may only be obtained from vehicle sensors which have limited range, resolution, and field of view. As a result, the information that the local planning module may obtain about its environment is highly dependent on the actions it evokes. Any vehicle motion will change the orientation of sensors, thereby enabling them to acquire different environmental details. Since the information needed to complete a plan is acquired in the process of plan execution, the local planning functions fit well within the stochastic planning framework.

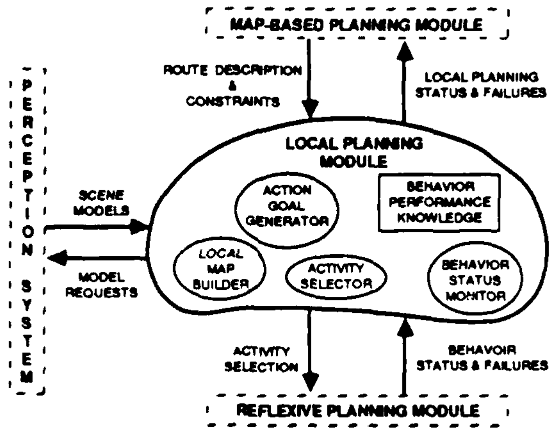

MAP-BASED PLANNING MODULE
ROUTE DESCRIPTION & CONSTRAINTS
LOCAL PLANNING STATUS & FAILURES
PERCEPTION SYSTEM
SCENE MODELS
MODEL REQUESTS
LOCAL PLANNING MODULE
ACTION GOAL GENERATOR
BEHAVIOR PERFORMANCE KNOWLEDGE
LOCAL MAP BUILDER
ACTIVITY SELECTOR
BEHAVIOR STATUS MONITOR
ACTIVITY SELECTION
BEHAVOIR STATUS & FAILURES
REFLEXIVE PLANNING MODULE

Figure 14. Functional elements of the local planning module.

The key to selecting appropriate activation sets is having the ability to make effective use of detailed knowledge of activity capabilities and limitations. Accordingly, as reflexive behaviors are defined and mixed together to form activities, these activities will be given symbolic declarative descriptions which can be interpreted by the local planning module. These descriptions will contain information such as:

- goals that would be satisfied,
- goals that would be thwarted,
- appropriate contexts,
- inappropriate contexts,
- limitations, and
- typical failures.

Use of this knowledge in conjunction with other data assimilated within the local planning module should permit effective selection of different activities for different situations.

4.2 DEVELOPMENT METHODOLOGY

In this section, we describe the methodology and tools we have developed in support of sensor-based planning. Most of the current development of the hierarchical planning system depends on having a diverse selection of robust virtual sensors and reflexive behaviors for controlling the ALV in a variety of situations. To develop these components, it has been necessary to provide a good realization of the tight control loop between sensing and action. While the best realization of this control loop may be found on the ALV itself. we have found that various simulation environments can provide more convenient and readily available development environments which are also more conducive to our rapid prototyping approach.

The real-world aspects of the ALV problem have forced the union of perception and planning technologies. In the past, image interpretation has been completely independent of planning issues. Perception had no purpose other than to "understand" a scene, and had no mechanism by which it could control the nature of any scenes made available to it. Similarly, planning work had assumed an idealized perception capability which did not require considering actions which could improve image quality. Realistic operating environments and real-time performance requirements have forced a shift of attention from complete analysis of a static world to focused and purposeful interpretation of a dynamically changing environment. Sensing and action must now be viewed as coordinated activities, each enhancing the abilities of the other. Every action has an impact on the ability to sense, and every sensory input has an impact on the ability to act. This new perspective on old problems has led to new methodologies for prototype development.

The methodology that we employ to create new virtual sensors and reflexive behaviors involves multiple stages of development as shown in Figure 15. This development plan is based on the idea of performing initial prototype development within very simple yet efficient simulation environments, then testing the more promising results in a detailed and time consuming simulation, and finally porting the most successful virtual sensors and reflexive behaviors to the ALV for validation. This methodology provides a mechanism to keep perception and planning work tightly integrated throughout the development process.

Initial prototype development, motivated by ALV mobility requirements begins with virtual sensor algorithm development by the perception group and reflexive behavior development by the planning group. This work is performed in parallel by the two groups, with constant exchange of ideas. The perception group analyzes image data sets obtained from the ALV to determine important features to be detected, and develop algorithms for detecting these features. These algorithms are eventually packaged as virtual sensors. Meanwhile, the planning group studies related control problems by imitating the available virtual sensors in a simplified simulation environment. As useful control schemes are developed, these are packaged as reflexive behaviors. Promising candidates from the prototypes are combined within an environment which simulates laser range scans and complex terrain in detail. Those techniques which are successful in this environment are then ready for testing on the ALV. These ALV experiments will be the ultimate litmus test for incorporation into the final system. Throughout this development process, failures will provide constant corrective input to the earlier development stages.

Each level of simulation in this approach supports a constant interplay between sensing and action. This allows all action plans to be formulated on the basis of partial information which can only be supplemented through further action. Communication pathways between components of planning and perception evolve as the algorithms and techniques evolve. As a result, a system built under this methodology will be geared to the real-time dynamic nature of the environment, and will ultimately exhibit more intelligent behavior than systems in which communication pathways between planning and perception are fixed at an early stage of he design.

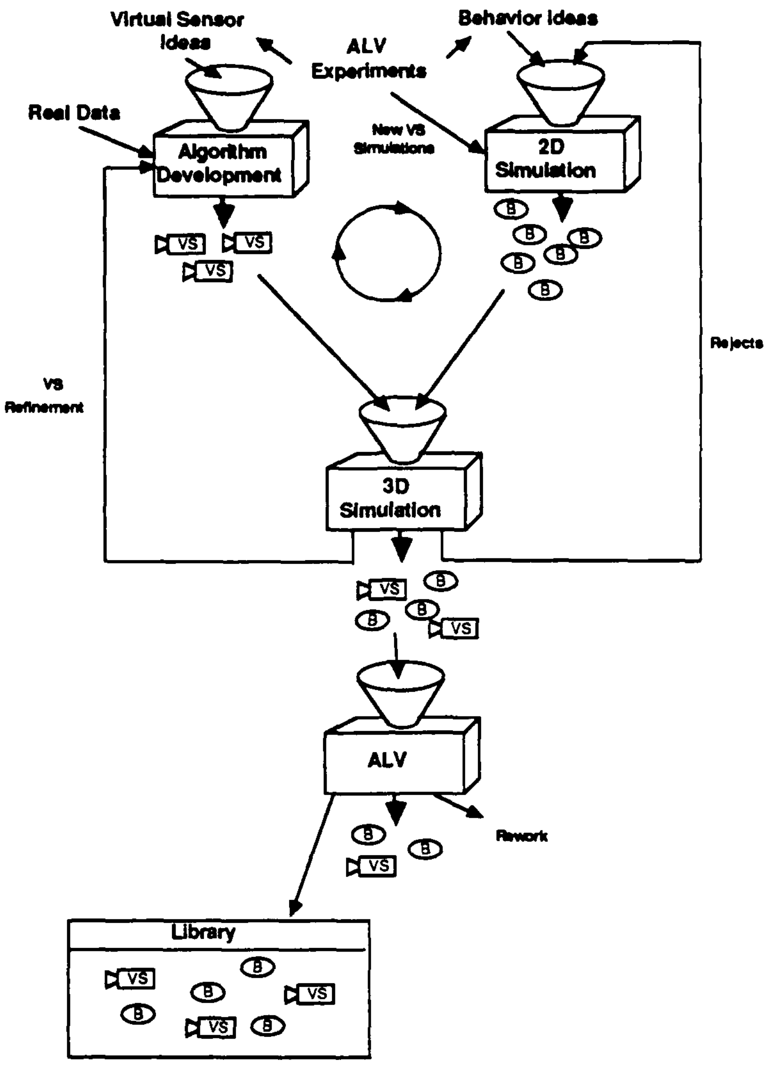


Figure 15. Development Cycle

4.3 PROGRESS

In this, our first year under contract, we have made progress in several key areas of sensor-based planning which we expect to impact future ALV demonstrations at Martin Marietta. Foremost among these accomplishments have been the development of two distinct ALV simulation environments, several new reflexive behavior and virtual sensor algorithms, and a map building capability for local planning. Also, in a joint effort with members of the Hughes ALV perception group, we have worked to formalize the dialogue between the perception and planning systems, and have developed a scene model language which can facilitate this dialogue (see Appendix I).

4.3.1 Simulation Environments

In this section we describe two software simulation environments which have been implemented for the purpose of facilitating the rapid development and testing of reflexive behaviors and virtual sensors. Both of these environments simulate the sensory inputs and physical actions of a vehicle which can move about in complex terrain.

While the key emphasis in both simulations is to close the loop between sensing and action, they differ significantly in terms of their level of detail and interactive performance. The first simulation environment we discuss is called "SERIM," an acronym for Simulated ERIM. This simulation provides a detailed three-dimensional simulation of terrain, using a fairly realistic simulation of an ERIM laser range scanner for sensory input. The second environment we discuss is referred to as a 2D simulation because it simulates the vehicle moving about on a flat plane with obstacles. This environment lacks the realism of SERIM, but has the advantage of providing a real-time vehicle simulation.

4.3.1.1 The SERIM System

The SERIM system is a simulation environment for testing the combination of virtual sensors produced by the perception group and reflexive behaviors produced by the planning group. The SERIM system includes a terrain modelling module, a simulated laser range scan, a simple vehicle simulation, and an interface to the reflexive planning module.

Terrain Modelling

One of the most important features of the SERIM test environment is that it supports the rapid creation of diverse terrain models. The terrain is represented by a grid of elevations, with associated grids representing other features, such as landcover or soil type. Although simple, this representation provides a flexible structure for creating complex terrain. A number of interactive tools have been developed to allow users to create these environments.

A typical terrain model employs an elevation grid to represent three-dimensional features such as hills, ditches, boulders, and roads. Just as in the ETL-Martin Marietta map database, variations in the elevation grid can represent the slopes of true terrain. Unlike the ETL database, however, this terrain model represents detailed terrain features with a resolution on the order of a few inches. To represent landcover and soil type information, additional grids are overlaid on the elevation data. This data can be used to help model differences in reflectance or spectral information. The combination of this information provides a versatile representation for outdoor terrain.

The creation and modification of terrain can be accomplished by a process similar to painting. The user starts with terrain that is perfectly flat. The user selects a texture of paint and uses a brush controlled by the mouse to paint on the terrain. The texture can have several features associated with it. The primary features of a texture are its color and the height. The color lets the user visualize what part of the terrain is being modified and usually represents a landcover type as well. The height is a specific height Ω. Wherever the texture is painted, the height Ω is added to the elevation grid (or subtracted if Ω is negative). Successive layers can be added by successive coats of paint. Another feature of a texture is its roughness μ. The roughness μ determines the random distribution of heights centered around the height Ω.

Painting one constant elevation cannot adequately produce terrain with slopes. Therefore a polygonal representation is used to model a slope. Each polygon represents a specific height. Polygons can be stacked onto each other forming a set of contours much like the contour lines in a topographic map. The grid and polygonal representations are merged by scan converting the polygons into the grid representation with heights between two polygonal sides computed by a linear interpolation between the two contour heights. Once the polygons have been converted, the grid heights produced by the painting process and the grid heights produced by the polygons are added together forming a composite elevation grid.

The Simulated Laser Range Scanner

To obtain realistic range imagery from our terrain models, we have developed a laser range scan simulation. This simulation generates range images from any perspective using a three-dimensional raytracing algorithm. Rays are traced in a scanning pattern much the same as actual laser rays are scanned in the ERIM laser scanner. For each pixel in the simulated range image, a ray is traced until it hits a point in the modelled terrain, yielding a corresponding range value. The resulting synthetic images are very similar to images taken with the ERIM scanner aboard the ALV.

One advantage of the laser scanner simulation is the flexibility it provides in changing the parameters of the sensor without any change in hardware. The SERIM simulation has several parameters that can be changed: **roll, pitch, yaw, resolution, field-of-view, scan distance,** and **ambiguity range**. Detailed descriptions of these parameters follow.

ROLL: The tilt from side to side of the scanner platform can be changed to any degree relative an ideal flat surface.

PITCH: The tilt from front to back of the scanner platform can be changed to any degree relative to an ideal flat surface.

YAW: The rotation from left to right can be changed to any degree relative to an absolute reference direction.

RESOLUTION: The delta change in angle between scan points can be changed (both vertical and horizontal).

FIELD-OF-VIEW: The scanner field of view can be changed (both vertical and horiziontal).

SCAN DISTANCE: The effective range of the scanner can be varied. This is the maximum distance that the scanner can see. This will help accommodate upgrades to the ERIM range scanner which will be gaining range from 100 feet to 100 meters.

AMBIGUITY RANGE: The range between ambiguity intervals can be adjusted to any distance.

In addition to the basic range scanning capabilities, we are also able to simulate scans of non-uniform surfaces of objects such as trees and bushes. To do this, we provide information in the landcover overlay grid to indicate these special types of surfaces. When the ray trace intersects with a piece of terrain marked with this landcover data, the distance returned may be specified by an arbitrary function of the actual distance. In the case of most brush, we add a random variance to the actual distance, making a smooth surface look arbitrarily rough.

The Vehicle Simulation

To support experimentation with vehicle control algorithms in the SERIM environment, a simple vehicle model has been implemented. This model provides a link between motion commands and the sensing capabilities of the simulated laser scanner. The vehicle model serves as a moving platform on which the laser scanner travels through the modeled environment according to planned vehicle actions. As the simulated vehicle moves through its environment, its orientation changes with the slope of the terrain, thereby effecting the position and orientation of the simulated scanner as well. As virtual sensors and reflexive behaviors process data from the simulated scanner to yield motion commands, this simulation closes the loop by causing the resulting motion to yield new sensory input.

The vehicle may be controlled by speed and turn rate commands from the reflexive planner, and has an independent process to move it through the SERIM environment accordingly. As the vehicle moves, its pitch and roll are altered by the surrounding terrain, causing the orientation of scanner to change as well. With a parameterized wheelbase, the pitch and roll are extracted from the average vector of four cross product vectors of four wheels to the center point of the vehicle. The control behaviors rely on other data in addition to range imagery, namely heading and location. Heading is obtained from a simulated compass which is modelled with a simple random error from the vehicle's true heading. The vehicle location is modelled by the summation of cumulative distance travelled, with random errors added along the way. These features provide a fairly realistic representation of the problems that might be encountered by the ALV.

Although we believe that a detailed simulation environment is necessary to provide validation of our virtual sensor and reflexive control techniques, we have found that the detailed simulation environment is much too cumbersome and slow to be useful for initial algorithm development. The primary problem is that proper ERIM scan simulation is much too slow for efficient experimentation. A typical run which might take three minutes in real time can take as much as thirty minutes to simulate. Since reflexive behavior develcpment requires repeated runs and revisions, this turn-around time is unacceptable. The SERIM simulation environment can only serve as a refinement tool for algorithms which are already fairly promising.

4.3.1.2 Two Dimensional Simulation

In an effort to provide an environment which supports rapid prototyping through real-time simulation of reflexive behaviors, we have developed a simplified two-dimensional simulation. Unlike the SERIM environment, all aspects of the two-dimensional environment are based on a planar approximation of the real world. Under this approximation, a vehicle is allowed to move in any direction on a plane so long as it does not collide with any obstacles placed on the surface of that plane. Although this simplification fails to capture some of the intricate planning problems related to maneuvering on steep slopes or bumpy terrain, many of the more basic planning problems such as obstacle avoidance and maneuvering through complex obstacle formations may be studied in an environment that allows efficient interaction.

To circumvent the turn-around delay of the SERIM environment without invalidating our development approach, we simulate the output of certain virtual sensors within the two-dimensional simulation. Currently, we have simulated an algorithm which processes ERIM data to provide range and extent data of free-standing obstacles. This simulation bypasses the ERIM simulation entirely by representing obstacles as objects from which appropriate extent data may be computed. This extent data is then used to control the vehicle on an otherwise flat surface with the primary objective being to avoid the observed obstacles.

Important aspects of vehicle motion are the vehicle shape and the means for vehicle control. The simulated vehicle is rectangularly shaped, making orientation an important planning issue. Control is maintained through the issuance of speed and turn-rate commands. With a speed and turn-rate provided, the vehicle will follow a fixed trajectory until otherwise commanded. By regularly updating the speed and turn rate commands, the planner can guide the vehicle in much the same way that a person guides a car through use of the steering wheel and accelerator pedal.

In order to place realistic time constraints on the planner, motion of the simulated vehicle is based on a real-time clock so that vehicle motion is not affected by processing time consumed by the planner. This way, the planner must consider its own reaction time when deciding how fast the vehicle should move. If the planner cannot react quickly enough, it must slow the vehicle down. Otherwise the vehicle could easily collide with an obstacle.

Figure 16 illustrates the two-dimensional simulation environment and the corresponding simulation of extents that would be detected by the obstacle-finding virtual sensor. Obstacles in the simulation are modeled as cylinders of variable height. The extents image is generated from the vehicle viewpoint just as if it were generated directly from an ERIM scan. As shown in the extents image, obstacle height and width are a function of perspective. This is identical to the way the extents computed from a real ERIM image appear.

A serious limitation of the two-dimensional simulation is that it ignores noise-related problems. In both real and simulated ERIM imagery, the extents virtual sensor often detects false obstacles due to noise. These noise-induced extents are usually near the upper boundary of the image where noise is most

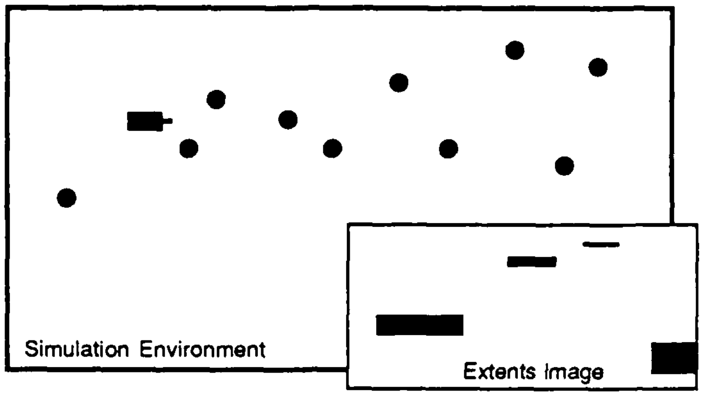


Figure 16. The two dimensional simulation environment

significant. Fortunately, this generally corresponds to objects that are furthest away from the vehicle, so it is expected that these errors will not significantly alter the performance of the operating reflexive behaviors. Nevertheless, while the two-dimensional environment is fine for initial development, we can never completely bypass SERIM for final verification.

4.3.2 Reflexive Behavior Development

In this section we describe a number of reflexive behavior algorithms which were developed to demonstrate the capabilities of the reflexive planning approach. We have developed some basic behaviors required for road following and cross-country obstacle avoidance. For obstacle avoidance, we have dealt with a number of considerations such as observability of obstacles, and finding obstacles along a vehicle's trajectory.

4.3.2.1 Simple Obstacle Avoidance

The reflexive behaviors for performing two simple activities WANDER and SEEK-HEADING, have both been implemented for control of the simulated vehicle. Figure 17 illustrates a sample path executed by the SEEK-HEADING activity, showing its ability to deal neatly with simple cul-de-sacs as well as other obstacles. In this path, one can see areas where an obstacle was detected by the simulated scan, causing the vehicle to slow and turn away. After turning away from an obstacle and finding open passage, the vehicle attempts to resume its heading to the right. Upon entering the cul-de-sac, the vehicle cannot find open passage despite its attempts to turn away from the obstacle wall. With repeated failures to find open passage, the direction of turn is maintained until the vehicle is free from the obstacle. The primary role that a higher level planner would have in this scenario would be to remember the cul-de-sac so that it could be avoided entirely in the future. Because it is composed of many of the same elements as SEEK-HEADING, the WANDER activity performs in much the same way, except that it does not attempt to orient itself toward a heading goal.

4.3.2.2 Basic Road Following

We have developed and, through simulation, have successfully demonstrated the ability to perfrom autonomous road following with asynchronous concurrent reflexive behaviors operating on test data from the Martin Marietta vision system. Although tests in completely artificial simulation environments have confirmed the viability of our reflexive control approach for a number of activities such as road following and obstacle avoidance, we felt it was necessary to demonstrate these capabilities in a more realistic environment. To obtain this realism without access to the actual vehicle, a simulation based on scene models from recorded ALV test runs was developed.

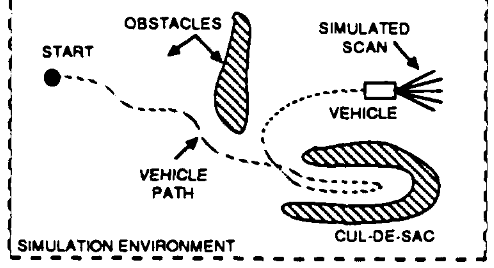


Figure 17. Sample vehicle path through the 2D simulation environment

Care was taken to ensure that the simulated perception output obtained from recorded data would appear no different to the planning system than it would if it were being acquired in real time. The recorded data consists of a sequence of scene models. One of these scene models is illustrated in Figure 18. As shown in the figure, a scene model consists of a series of points for the right and left side of the perceived road. and a point indicating where the vehicle was located when the scene was acquired. Also recorded with each scene model is the time elapsed from the moment of scene acquisition to the moment the scene model was completed. This represents the scene model generation delay. To create an authentic simulation with this data, it is necessary to account for both scene acquisition and scene model generation. Proper scene acquisition simulation requires that no scene may be available for model generation until the vehicle has passed the recorded acquisition location. Once this has happened, scene model generation may be simulated by imposing a delay on the availability of the data to the planner that corresponds to the scene model generation delay associated with the acquired scene.

Simplified vehicle dynamics have also been incorporated into the simulation to account for the effects that response delays will have on control stability. Currently, limits may be placed on both the linear and the angular acceleration of the vehicle. With these limits, the response to commands for sudden changes in speed and turn rate will be delayed.

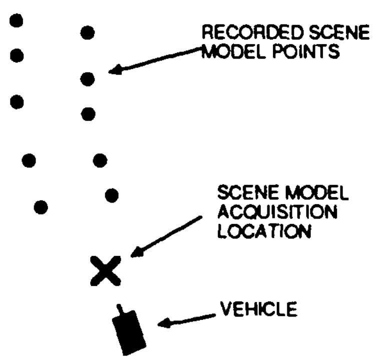


Figure 18. Simulation with recorded scene model data

Road following in this simulation was performed by the concurrent operation of two distinct reflexive behaviors. One behavior was concerned only with controlling vehicle speed, simply making the speed proportional to a measure of scene quality and the distance to the end of the currently available scene. The other behavior was concerned only with controlling the rate of turn for the vehicle, making it proportional to the scene quality and a heading error measure. The formulas for these behaviors are shown in Figure 19.

$$\text{SPEED} = g1 * (1 + \text{scene-quality}) * \log(\text{distance-to-scene-end})$$

$$\text{TURN-RATE} = g2 * \text{scene-quality} * \text{heading-error}$$

$$\text{distance-to-scene-end} = \hat{h} \cdot \hat{d}$$

$$\text{scene-quality} = \frac{(1 + \text{Li-max}) \cdot (1 + \text{Ri-max})}{(1 + \text{Li-max}) + (1 + \text{Ri-max})}$$

$$\text{heading-error} = \frac{\sum_{i}^{0 \text{ to } i} \left(C \cdot \hat{X}_{Li} \times \hat{h} \cdot \hat{X}_{Li} \cdot \hat{h} - \text{offset} + C \cdot \hat{X}_{Ri} \times \hat{h} \cdot \hat{X}_{Ri} \cdot \hat{h} + \text{offset} \right)}{\text{Li-max} + \text{Ri-max}}$$

$$C = 1 \text{ for } \hat{X} \cdot \hat{h} > 0, \ -0.1 \text{ otherwise}$$

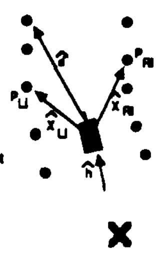


Figure 19. Formulas for reflexive road following behaviors

In these formulas, scene quality is determined by the heuristic that scene models with an equal number of points on the left and right edges tend to be the most reliable. The *Li-max* and *Ri-max* terms in the scene quality equation stand for the total number of points for the left and right edges respectively. The heading error term is computed from the cross products of the vehicle heading vector with vectors from the vehicle to each of the road points. Each of these cross products yields a term which is equivalent to the offset of the vehicle from the side of the road. Also, a higher weight is given to offsets from points farther ahead of the vehicle and a lower weight is given to all points behind the vehicle.

Using the above algorithms in the simulation environment, we have obtained encouraging performance results. Even with fairly moderate gain settings, providing a good margin for stability, the simulated vehicle has averaged over 9 Km/hour on data that was acquired from a run that averaged only 3.8 Km/hour. Martin Marietta has confirmed that they too could have run faster on the same data, but instead, were operating at a conservative fixed speed. The importance of these results is not to show that we have developed a superior control algorithm, but rather, to demonstrate that concurrent reflexive behaviors can be used to provide effective vehicle control. Our architecture dictates that many such reflexive behaviors must be developed to handle a wide variety of situations, so the algorithms described above are only a small contribution to our growing library of behaviors.

4.3.2.3 Obstacles in the Vehicle's Path

We have developed specialized behaviors and a virtual sensor for detecting and avoiding obstacles in a curved path. Previously we had a virtual sensor for detecting the nearest obstacle directly in front of an autonomous vehicle. Behaviors built with this virtual sensor could only react to obstacles in front of the vehicle. We encountered a problem, however, when the vehicle was placed in an area densely populated with obstacles. When the vehicle was moving along, it would detect an obstacle in front of it, slow down, and then turn away as was expected. However, because the virtual sensor was developed to detect obstacles in front of the vehicle, it could not react to obstacles to the side when it was turning. Thus many times the vehicle would collide with an obstacle to the side when it was turning.

To solve this problem we developed a virtual sensor that detects obstacles in the direction that the vehicle is turning. This virtual sensor, called NEAREST OBSTACLE IN CURVED PATH, first computes R_t, the turn radius of the vehicle, from speed and turn rate δ_θ: $R_t = speed/\delta_\theta$. (See Figure 20) The turn-center for the vehicle, T_c, is then computed from the vehicle's current position and R_t. Next the left and right path edges are calculated from T_c, R_t, and the clearance for the vehicle. The sensor then scans the object extents of obstacles in the current scene for obstacles that would intersect the curved path trajectory. When the nearest obstacle intersecting the curved path trajectory has been found, the virtual sensor returns the distances to the left and right extents of the obstacle as well as the offsets of the left and right extents to the center of the curved path trajectory.

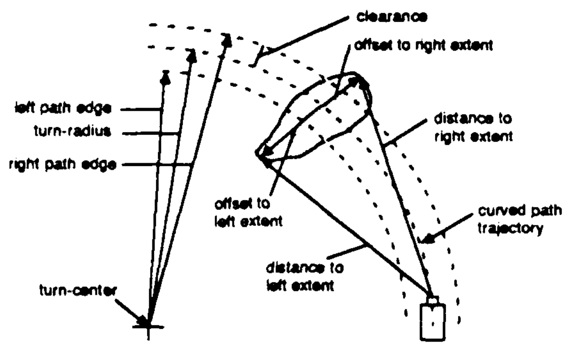


Figure 20. Detecting an obstacle in a curved path trajectory

Two behaviors, *SLOW FOR OBSTACLE IN CURVED PATH* (SFOCP) and *TURN FOR OBSTACLE IN CURVED PATH* (TFOCP), have been developed that utilize the virtual sensor *NEAREST OBSTACLE IN CURVED PATH*. As their names imply, SFOCP slows the vehicle when the virtual sensor has detected an obstacle intersecting its curved path trajectory, and TFOCP uses the left and right offsets provided by the virtual sensor to turn the vehicle away from the obstacle. An activity called CURVED-PATH-WANDER was defined, comprising the behaviors SFOCP and TFOCP. This activity was tested in the 2-D simulation environment consisting of a dense scattering of obstacles. The developed behaviors and virtual sensor for detecting and avoiding obstacles in a curved path trajectory did a better job at navigating through the obstacles than the previous behaviors and virtual sensor that could only detect obstacles directly in front of the vehicle

5 CONCLUSION

We have presented an architecture for autonomous vehicle planning and have provided the justification for its design. We believe in the long run that it would be advantageous for Martin Marietta to adopt this architecture. We recognize that in order for this to happen it will be necessary for us to provide suitable demonstrations of our architecture on the ALV. Our goals for 1987 are focused around a number of demonstration objectives.

5.1 KEY ISSUES

Our primary goal in developing an autonomous vehicle planning system is to ultimately impact demonstration requirements for the Martin Marietta ALV. In order to achieve this goal we must address a number of technical problems which are critical to porting our architectural concepts to Martin Marietta. The primary issues involve developing appropriate hardware and software interfaces, developing and implementing new virtual sensors and behaviors appropriate for vehicle hardware, and obtaining better sensing and maneuvering capabilities for the ALV.

5.1.1 Interface Development

The primary objective of interface development is to provide a convenient and efficient link between our existing Symbolics-based planning software and the real-time system architecture of the ALV. The key link in this interface is the Pronet network and software protocols. By developing a link between Symbolics and the Pronet network we will be able to incorporate our architecture directly into the ALV hardware.

We will be developing the low level driver software to support the Pronet interface on Symbolics hardware. This software will duplicate the functionality of the driver software that has been written by Martin for the SUN and INTEL processors. We will be able to test this interface at our site using available Symbolics and SUN processors.

On top of the low-level driver software, we will be implementing higher-level software protocols to support the perception/planning interfaces needed by our architecture. Specifically, we will be developing message passing protocols along with high-speed data stream protocols. The message passing will support low-bandwidth communication between higher levels of perception and planning while the data stream protocols will support high-bandwidth communication between virtual sensors and reflexive behaviors. The proper implementation of these protocols should greatly simplify the process of porting our planning software to Martin.

5.1.2 Virtual Sensor and Behavior Development

To support cross-country demonstrations with the ALV, we will need to develop a number of virtual sensors and reflexive behaviors. Specific capabilities that must be supported are: finding road entry and exit points; avoiding cross-country obstacles; maneuvering in slippery terrain; and detecting, following, and crossing ravines.

The basic problem with road entry and exit is that most roads have very few areas where entry and exit is possible. Typically, small gullies and embankments bordering roads make entry and exit difficult. Unfortunately, the ETL-Martin Marietta map data does not include information about these important features, making map-based planning about road entry and exit infeasible. As a result, we will have to rely heavily on perception capabilities for searching for viable entry and exit areas. Where viable areas are found, planning will have to determine the appropriate method for passage based on the nature of the passable area.

Cross-country obstacle avoidance can be either complex or simple. Simple situations involve isolated obstacles on relatively flat terrain.

In complex situations, obstacles and terrain features combine to create difficult maneuverability problems. The slope and type of terrain that obstacles are on may cause failure of techniques which are otherwise effective in simple situations. For example, obstacles on a steep slippery slope may not be avoided in the same way as they would be on flat terrain. A variety of strategies which depend on the characteristics of the terrain will be developed to solve these problems.

From observation of vehicle performance in slippery terrain, we feel that special maneuvering capabilities will be needed to prevent the vehicle from getting stuck. Inflexible control algorithms are incapable of recognizing and coping with failure. We will be developing techniques for detecting failure due to slippage and recovering with special maneuvers such as rocking the vehicle to get out of a rut.

Since ravines form some of the most significant obstructions in the Martin terrain, we will be devoting special attention to maneuvering around these features. We must consider ravines from two perspectives. We must primarily be able to avoid them, but we also may be able to exploit them for navigation. Avoidance of ravines requires either skirting around them, or finding areas where they may be crossed. Since ravines are known map features, we may also use them for navigation by following them as a means of staying oriented. Behaviors will be developed that exploit the characteristics of ravines to perform these tasks.

5.1.3 Vehicle Capability Requirements

To to deal with the many cross-country mobility problems, several enhancements to the ALV sensing capabilities will be needed. Namely, sensors will be needed to help detect slippage, to correct the LNS when slippage occurs, to properly register motion when the vehicle backs up, to detect chassis pitch and roll relative to the wheelbase and to level ground. We will be encouraging Martin Marietta to provide these capabilities.

5.2 EXPERIMENTAL PLANS

We have generated a detailed outline of our experimental plans for demonstrating our planning software on the Martin vehicle. Because our planning architecture is primarily divided into the three layers of map-planning, local planning, and reflexive planning, our experiments are intended to test each of these layers individually before the integrated system is tested.

The tables included at the end of this section summarize our experimental plans and objectives. In these plans, we have segregated near-term goals from longer-term goals. A brief discussion of the motivation behind these experiments is presented in the paragraphs that follow.

5.2.1 Map Planning Experiments.

The primary objective of the map planning experiments is to demonstrate and test the validity of our existing map planning techniques. Since we wish to do this before all the issues of sensor-based planning have been solved, the experiments are designed to be performed with human operators assisting in the execution of planned routes. We intend to test both cross-country and road network route planning capabilities.

For cross-country planning experiments, we will have the route planner generate a path in terms of a series of points in world coordinates. With the aid of a human operator for avoiding obstacles, we hope to be able to have the Martin vehicle traverse this route. Should this not be possible, we will at least travel the planned route with a four-wheel-drive vehicle in order to make visual observations of terrain difficulty. To enhance these tests, we also intend to plan a few routes which have intermediate goals of reaching good vantage points for observing landmarks.

For road network experiments, we will simply drive the planned routes in standard vehicles. The primary purpose for travelling these routes will be to see if there are particular features along the roads and their intersections which may cause potential difficulty for perception algorithms. Also, we will be able to compare planned routes with routes that people have planned to see if they agree.

Later, in 1987, we expect to demonstrate replanning capabilities. To do this without having all the necessary low-level perception capabilities, we will perform the same operator-assisted cross-country traversal tasks as described above with the addition of having the operator signal plan failure to the map planner when extremely difficult terrain is encountered. At that time, the map planner would be required to replan a new route from the failure location to the goal.

5.2.2 Reflexive Planning Experiments

The primary objective of the reflexive planning experiments is to validate our reflexive planning methodology, and begin to develop a repertoire of useful reflexive behaviors. We feel the validation of our reflexive planning methodology is essential because this constitutes the lowest level of our planning hierarchy and has a significant influence on how the higher levels must operate. At this point in time, we see the approach that Martin is taking diverging from ours to such an extent that we feel we may not easily be able to integrate our techniques into their system. Given Martin's relative success at satisfying near-term milestone objectives, they are justifiably reluctant to spend a great deal of time and effort attempting to integrate unproven technology. The existing approach, however, does not appear to be highly amenable to the types of higher level control which we feel will be important in the more complex demonstrations in 1987 and '88. We therefore feel it is necessary for us to make convincing demonstrations that our approach can perform as well if not better than theirs so that we may pave the way to integrating our entire hierarchical planning system onto their vehicle architecture. In the process of developing these demonstrations, we will also be starting to build the required repertoire of reflexive behaviors needed to meet future ALV milestones.

5.2.3 Local Planning Experiments

The purpose of the local planning experiments is to develop and demonstrate local maneuvering capabilities that cannot be achieved by reflexive behaviors alone. Under our reflexive control approach, the local planning module is required to use assimilated perceptual data to switch activation of different groups of reflexive behaviors to suit traversal goals. This means that creation of a suitable repertoire of reflexive behaviors will be a prerequisite to experimentation with the local planning module on the Martin vehicle. Because of this requirement, we do not expect this experimentation to take place before late 1987 or early '88. Typical situations where we expect local planning capabilities to be needed are for handling transitions such as asphalt to dirt roads, going through road intersections, and getting on and off roads. We also expect the local map building capabilities of the local planner to assist with problems such as getting out of dead-ends and cul-de-sacs.

5.2.4 Integrated System Experiments

In the 1987, '88, and '89 timeframe, we expect to start tying the map-based, local, and reflexive planning modules together into a complete planning system. This should allow the achievement of several critical ALV milestones. The first milestone we hope to demonstrate is the planning and execution of a simple cross-country route. This will involve limited capabilities from each of the three modules. Other experiments, oriented toward preparation for later demonstrations, will include combining on-road and off-road navigation techniques, planning to go to specified landmarks, and planning and executing cross-country traversal in both simple and complex terrain. Each of these experiments will allow further development of the interfaces between the different planning layers.

REFERENCES

J.S.B. Mitchell, "Planning Shortest Paths," PhD Thesis, Department of Operations Research, Stanford University, August, 1986. (Available as Research Report 561, Artificial Intelligence Series, No. 1, Hughes Research Laboratories, Malibu, CA.)

J.S.B. Mitchell, D.W. Payton, and D.M. Keirsey, "Planning and Reasoning for Autonomous Vehicle Control," To appear in International Journal for Intelligent Systems, John Wiley & Sons,1987.

J. L. Crowley, "Navigation for an Intelligent Mobile Robot," IEEE Journal of Robotics and Automation, RA-1, pp. 31-41, March 1985.

H. P. Moravec, "The Stanford Cart and the CMU Rover," Proceedings of the IEEE, 71, pp. 872-884, July 1983.

N. J. Nilsson, "Shakey the Robot," SRI AI Center Technical Note 323, April 1984.

R. A. Brooks, "A Layered Intelligent Control System for a Mobile Robot," ISRR; Third International Symposium of Robotics Research, Gouvieux, France, October 7-11, 1985.

S. Y. Harmon, "USMC Ground Surveillance Robot (GSR) : A Testbed for Autonomous Vehicle Research," Proceedings of the Fourth University of Alabama Robotics Conference, Huntsville, Alabama, April 24-26, 1984.

R. Wallace, A. Stentz, C. Thorpe, H. Moravec, W. Whittaker, T. Kanade, "First Results in Robot Road-Following," Ninth International Joint Conference on Artificial Intelligence, Los Angeles, California, pp. 1089-1095, August 18-23, 1985.

R. Chavez and A. Meystel, "Structure of Intelligence for an Autonomous Vehicle," IEEE International Conference on Robotics, Altanta, Georgia, March, 1984.

C. Isik and A. Meystel, "Decision Making at a Level of a Hierarchical Control for Unmanned Robot," IEEE International Conference on Robotics and Automation, San Franscisco, California, April 7-10, 1986.

G. Giralt, R. Sobek, and R. Chatila, "Multi-level Planning and Navigation System For a Mobile Robot: A First Approach to Hilare," Proc. of IJCAI-79, Vol 1, Tokyo, 1979.

D. Keirsey and J.S.B. Mitchell, "Planning Strategic Paths thru Variable Terrain," SPIE Conference on Applications of Artificial Intelligence, Arlington, Virginia, April 29-May 4,1984.

APPENDIX I

STRAWMAN INTERFACE MODEL DEFINITIONS

by

Members of the Perception and Reasoning Groups
Hughes AI Center

Mike Daily, Rina Dechter, Dave Keirsey, Joe Mitchell,
Karen Olin, Dave Payton, Kurt Reiser, Teresa Silberberg,
Felicia Vilnrotter, and Vincent Wong

November, 1985

A strawman set of model definitions to be used in the interface between Reasoning and Perception subsystems have been defined. These models are intended to provide a language framework through which perception and reasoning systems may communicate. A reasoning system will use the models to clearly communicate what it needs from perception, while a perception system will use the models to commmunicate what it is "seeing." The models provide the means by which information is formatted such that it can be understood by both reasoning and perception. It is hoped that these model definitions will give working group members a concrete framework in which to discuss their own concepts and representations. It must be emphasized that these models *DO NOT* attempt to define any of the models or data structures that may be used *internally* by either perception or reasoning.

We have chosen to use FLAVORS as the definition language for these models because it is a convenient, well documented, and adequately expressive language for communicating the necessary concepts. This, however, should not be viewed as a statement in favor of FLAVORS as the ultimate definition language for interface models. It may eventually be found necessary to construct a special definition language to effieciently support all the functionality needed from these models.

FLAVORS allow the definition of class/subclass object hierarchies. In these hierarchies, subclasses are specializations of their superclass. This means that they may inherit all of the attributes of their superclass while adding a few more of their own. For example, the simple hierarchy shown in Figure-1a may be expressed in FLAVORS as shown in Figure-1b. Note that since CONIFER and DECIDUOUS-TREE inherit the attributes or slots from TREE, height and trunk do not have to be named explicitly in their definitions.

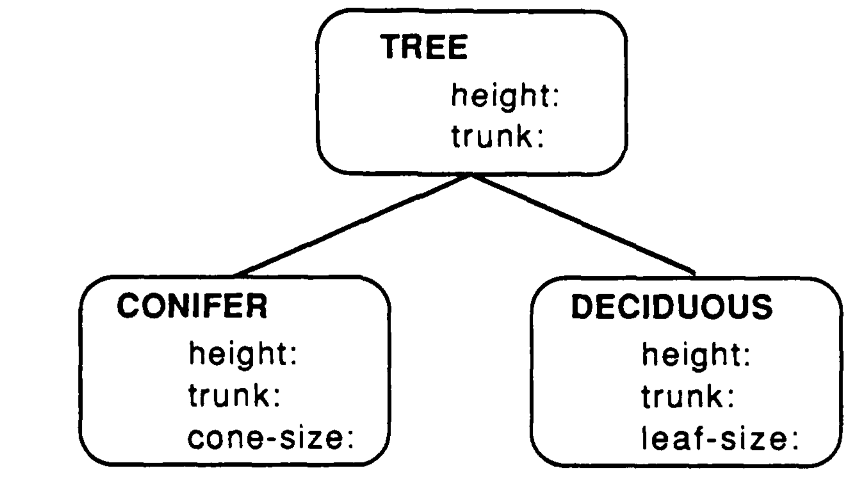


Figure 1a. Sample Model

```
(defflavor tree
      ((height)
       (trunk))
      ())

(defflavor conifer
      ((cone-size))
      (tree))

(defflavor deciduous
      ((leaf-size))
      (tree))
```

Figure 1-b. Sample Model in FLAVORS

The syntax of the flavor definition is as follows:

```
(defflavor <name>
      ((<attribute-name> <default-value>)
       (<attribute-name> <default-value>)
      ...)
      (<superclass> <superclass> ...))
```

Since an object can have more than a single superclass, class hierarchies often form a graph rather than a simple tree structure.

In our model definitions, we use the <default-value> of each FLAVOR slot to constrain the type and number of objects that may fill those slots. A typical <default-value> entry would be:

```
(:ALLOWED-MODELS (region profile) :CARDINALITY (>= 2))
```

This says that the slot is restricted to be filled with two or more instances of REGION and/or PROFILE models. If :CARDINALITY is omitted, it is assumed to be 1. In this way, slots of complex models are defined in terms of other simpler models until we finally arrive at the most basic primitives.

When the reasoning system makes a request to perception, it asks for a selected model to be instantiated. Along with this request, reasoning indicates which attributes of the model it wishes to have instantiated, and at the same time, reasoning may further constrain the options on how the slots may be filled. Furthermore, since model slots are filled with instances of other models, reasoning must also select which attributes should be instantiated within these models. Thus, any single request actually specifies which slots are to be filled for models within models within models etc. It seems likely that we might eventually arrive at a set of typical requests that we can simply call by name rather than specifying the exact attributes needed at each level of the model structure.

Perception examines a request to determine which features of the requested model it must provide and what representations for these features it should use. Perception prepares a reply for reasoning by building a model structure that is identical to the request model, but with the appropriate slots filled with instances of perceived objects. These objects now provide common ground through which reasoning may request more information.

It is expected that some of the models presented hereunder may eventually be found to be unnecessary or excessively detailed whi'~ others may become greatly enhanced. Furthermore, not all of the lowest level models have been defined. It is hoped that contributions from other working group members will help to complete these model descriptions.

Fundamental Mixin Definitions

Some information must be fundamental to all models. This primarily involves a means for communicating how any model should be interpreted First, any model may be interpreted from a number of reference frames including:

Map -- this may be ETL/UTM or any other grid-based reference
Vehicle
Image-Plane
Object-Centered

In addition, all models have been defined for multiple dimensionality characteristics including 3D, 2D, and 1D. These fundamentals are established with the following *mixin* s.

```
(defflavor viewpoint-mixin
        ;; This is needed in every model to establish the way in which
        ;; perception should interpret the model.
        ((reference-frame
            (:ALLOWED-MODELS
                (map vehicle image-plane object-centered)))
         (dimensionality  (:ALLOWED-MODELS (3D 2D 1D)))
         (fov (:ALLOWED-MODELS (field-of-view)))
         (expected-spatial-orientation
            (:ALLOWED-MODELS (orientation)))
         (expected-distance-from-viewer
            (:ALLOWED-MODELS (length-measure)))
         (expected-absolute-placement
            (:ALLOWED-MODELS (point))))
        ())

(defflavor field-of-view
        ;; There are two ways to look at field of view. One is to consider a
        ;; rectangle on the image plane, the other is to consider a left and
        ;; right azimuth and a top and bottom elevation angle.
        ((frame-center (:ALLOWED-MODELS (unit-vector)))
         (frame-size (:ALLOWED-MODELS (rectangle)))
         (azimuth-elevation-range
          (:ALLOWED-MODELS (azimuth-elevation-range))))
      ())
```

(defflavor azimuth-elevation-range

```
      ((azimuth-range
          (:ALLOWED-MODELS (angle) :CARDINALITY 2))
       (elevation-range
          (:ALLOWED-MODELS (angle) :CARDINALITY 2)))
      ())
```

(defflavor orientation

```
      ;; For any rotational transformation, we rotate around the
      ;; minor-axis first and then the major-axis.
      ;; (This should be equivalent to pitch and roll.)
      ((rotation-on-minor-axis (:ALLOWED-MODELS (angle)))
       (rotation-on-major-axis (:ALLOWED-MODELS (angle))))
      ())
```

(defflavor request-mixin

```
      ;; This contains all the information that is specific to any request.
      ;; All of these slots (including the ones inherited from
      ;; viewpoint-mixin) must be specified by the Reasoning system when
      ;; it makes a request to Perception.
      ((confidence-method)          ;Models fo, these slots still
       (resolution)                 ;need to be defined.
       (priority))
      (viewpoint-mixin))
```

(defflavor reply-mixin

```
      ;; This contains the base information that is specific to any reply.
      ;; All of these slots must be filled by the Perception system when it
      ;; replies to Planning.
      ((timestamp)
       (confidence-value))
      ())
```

ROAD and INTERSECTION Model Definitions.

```
(defflavor road
        ((edge1 (:ALLOWED-MODELS (road-edge)))
         (edge2  (:ALLOWED-MODELS (road-edge)))
         (road-region (:ALLOWED-MODELS (region)))
         (side1 (:ALLOWED-MODELS (roadside)))
         (side2 (:ALLOWED-MODELS (roadside)))
         (centerline (:ALLOWED-MODELS (edge)))
         (intersection (:ALLOWED-MODELS (intersection))))
        (request-mixin reply-mixin))

(defflavor road-edge
        ;; You need to be able to identify road edges as either the left/right
        ;; or the near/far because we want the road model to be independent
        ;; of viewpoint. This means that we want the same road model for
        ;; roads you are traveling on as for roads you are observing or
        ;; approaching from a distance. Orientation is relative to the other
        ;; road-edge.
        ((orientation (:ALLOWED-MODELS (left right near far)))
         (edge (:ALLOWED-MODELS (edge))))
        (request-mixin reply-mixin))

(defflavor dirt-road
        ()
        (road))

(defflavor asphalt-road
        ()
        (road))

(defflavor wagon-rut-road
        ((center-strip (:ALLOWED-MODELS (region cross-section)))
         (left-rut (:ALLOWED-MODELS (region)))
         (right-rut (:ALLOWED-MODELS (region))))
        (road))


(defflavor roadside
        ;; We use the term "roadside" because it applies in the more general
        ;; case when you don't really have a distinct "road shoulder".
        ;; This is orientation relative to the other roadside.
        ((orientation (:ALLOWED-MODELS (left right near far)))
         (off-road-description (:ALLOWED-MODELS (scene)))
```

```
  ;; This might be supplied by some kind of tracking algorithm.
  (change-in-distance-to-vehicle-over-time
     (:ALLOWED-MODELS (length-measure))))
 (request-mixin reply-mixin))
```

(defflavor edge

```
 ((direction (:ALLOWED-MODELS (vector heading)))
  (shape (:ALLOWED-MODELS (curve)))
  (width (:ALLOWED-MODELS (length-measure)))
  ;; This might be supplieu by some kind of tracking algorithm.
  (change-in-distance-to-vehicle-over-time
     (:ALLOWED-MODELS (length-measure))))
 (request-mixin reply-mixin))
```

(defflavor intersection

```
 ;; This intersection model views each road coming out of the
 ;; intersection as a distinct object.  Each road-pair expresses the
 ;; angle between two of the roads leaving the intersection.  A single
 ;; road may be included in more than one road-pair descriptor.
 ((road-pairs (:ALLOWED-MODELS (road-pair) :CARDINALITY (> 1)))
  (center-point (:ALLOWED-MODELS (point))))
 (request-mixin reply-mixin))
```

(defflavor road-pair

```
 ((angle (:ALLOWED-MODELS (angle)))
  (road1 (:ALLOWED-MODELS (road)))
  (road2 (:ALLOWED-MODELS (road))))
 (request-mixin reply-mixin))
```

The following three intersection models have a slightly different interpretation of what distinct roads are. Here, if a road continues in the same direction, the parts before and after the intersection are both united into a single object.

(defflavor T-intersection

```
 ((continuing-road (:ALLOWED-MODELS (road)))
  (ending-road (:ALLOWED-MODELS (road))))
 (intersection))
```

(defflavor Y-intersection

```
 ((base-road (:ALLOWED-MODELS (road)))
  (left-branch (:ALLOWED-MODELS (road)))
  (right-branch (:ALLOWED-MODELS (road))))
 (intersection))
```

```
(defflavor X-intersection
        ((road-1 (:ALLOWED-MODELS (road)))
         (road-2 (:ALLOWED-MODELS (road))))
        (intersection))
```

OBSTACLE Model Definitions

Obstacles may be either bounded or unbounded. The idea here, is that some obstacles such as rocks, trash cans, or traffic cones may have clearly defined boundaries while other obstacles such as ravines have much fuzzier boundaries and cannot necessarily be dealt with in the same way.

```
(defflavor obstacle
        ((orientation (:ALLOWED-MODELS (orientation)))
         (shape (:ALLOWED-MODELS (object-shape)))
         (containing-object (:ALLOWED-MODELS (road scene))))
        (request-mixin reply-mixin))

(defflavor bounded-obstacle
        ((height (:ALLOWED-MODELS (length-measure)))
         (width (:ALLOWED-MODELS (length-measure))))
        (obstacle))

(defflavor unbounded-obstacle
        ;; Sometimes the best representation might just be the average
        ;; surface normal for the object.  This would at least give the
        ;; planner some idea of its potential traversability.
        ((average-normal (:ALLOWED-MODELS (surface-normal))))
        (obstacle))

(defflavor ravine
        ()
        (unbounded-obstacle))

(defflavor rock
        ()
        (bounded-obstacle))

(defflavor road-obstacle
        ((distance-from-left-edge
            (:ALLOWED-MODELS (length-measure)))
         (distance-from-right-edge
            (:ALLOWED-MODELS (length-measure))))
```

```
    (bounded-obstacle))
```

(defflavor object-shape
```
    ((profile (:ALLOWED-MODELS (cross-section)))
     (3D-surface-approximation (:ALLOWED-MODELS (region)))
     (medial-axis (:ALLOWED-MODELS (medial-axis)))
     (angular-extent (:ALLOWED-MODELS (angular-extent)))
     (leading-edge (:ALLOWED-MODELS (cross-section))))
    (request-mixin reply-mixin))
```

SCENE Model Definitions

The SCENE model attempts to encompass all general models of space in the vehicle environment. Two distinct views of space are possible, that of a set of open corridors i.e. FREE-SPACE or that of a set of obstructions i.e. OBSTACLE-SPACE.

(defflavor scene
```
    ((scene-map (:ALLOWED-MODELS (subdivided-region)))
     (cross-section (:ALLOWED-MODELS (cross-section))))
    (request-mixin reply-mixin))
```

(defflavor free-space
```
    ((corridors
        (:ALLOWED-MODELS (disconnected-region connected-region)))
     (open-sectors
        (:ALLOWED-MODELS (angular-extent) :CARDINALITY (>= 1))))
    (scene))
```

(defflavor obstacle-space
```
    ((obstacles
        (:ALLOWED-MODELS (obstacle) :CARDINALITY (>= 0)))
     (blocked-areas
        (:ALLOWED-MODELS  (disconnected-region connected-region))))
    (scene))
```

LANDMARK Model Definitions

A great deal more work is needed here. Some landmarks may also be considered obstacles.

```
(defflavor landmark
        ((orientation (:ALLOWED-MODELS (orientation)))
         (shape (:ALLOWED-MODELS (object-shape)))
         (containing-object (:ALLOWED-MODELS (scene))))
        (request-mixin reply-mixin))

(defflavor telephone-pole
        ((height (:ALLOWED-MODELS (length-measure))))
        (landmark obstacle))

(defflavor power-tower
        ((height (:ALLOWED-MODELS (length-measure))))
        (landmark obstacle))
```

REGION Model Definitions

The concept of REGION attempts to encompass segments of space of any dimensionality. Only the subclasses of region form useful representations. Subdivided regions can be represented solely by a collection of interior subregions. However, it may also be useful to explicitly know the outer boundary of the region being subdivided.

```
(defflavor region
        ((descriptors
           (:ALLOWED-MODELS (region-descriptor) :CARDINALITY (>= 1))))
        (request-mixin reply-mixin))

(defflavor connected-region
        ((boundary-representation (:ALLOWED-MODELS (closed-boundary)))
         (interior-representation (:ALLOWED-MODELS (interior))))
        (region))

(defflavor disconnected-region
        ((component-regions
           (:ALLOWED-MODELS (connected-region) :CARDINALITY (> 1))))
        (region))
```

```
(defflavor subdivided-region
        ((subregions (:ALLOWED-MODELS (region) :CARDINALITY (>= 1)))
         (outer-boundary (:ALLOWED-MODELS (closed-boundary))))
        (region))

(defflavor region-descriptor
        ;; Region descriptors are for associating attributes with regions
        ((color (:ALLOWED-MODELS (color)))
         (surface-normal (:ALLOWED-MODELS (surface-normal))))
        (request-mixin reply-mixin))

(defflavor sky
        ((condition (:ALLOWED-MODELS (clear cloudy overcast))))
        (region-descriptor))
```

APPENDIX II

This appendix illustrates the modes of interaction between perception and planning. Starting with system initialization messages, the planner asks perception to look for a road, and eventually, to expect to see a building. Once the road is detected, the planner specifies virtual sensor groups which will be used for road following. Once perception has approved these groups, the planner tells perception to invoke a virtual sensor group and it activates a selected group of behaviors. After system initialization, virtual sensors, behaviors, and other peception and planning processes operate concurrently to navingate the vehicle. The charts show a possible road following scenario, indicating the responses of each process over time.

System Initialization

PLANNER | **PERCEPTION**

(Vehicle @Start)

Plan Path

Generate Path Description

$\longrightarrow$

Provide Road: FOV:forward REP: region/edge

$\longrightarrow$

Demon Detect: Building FOV: forward REP: vector

Model Road

Select Building Detection

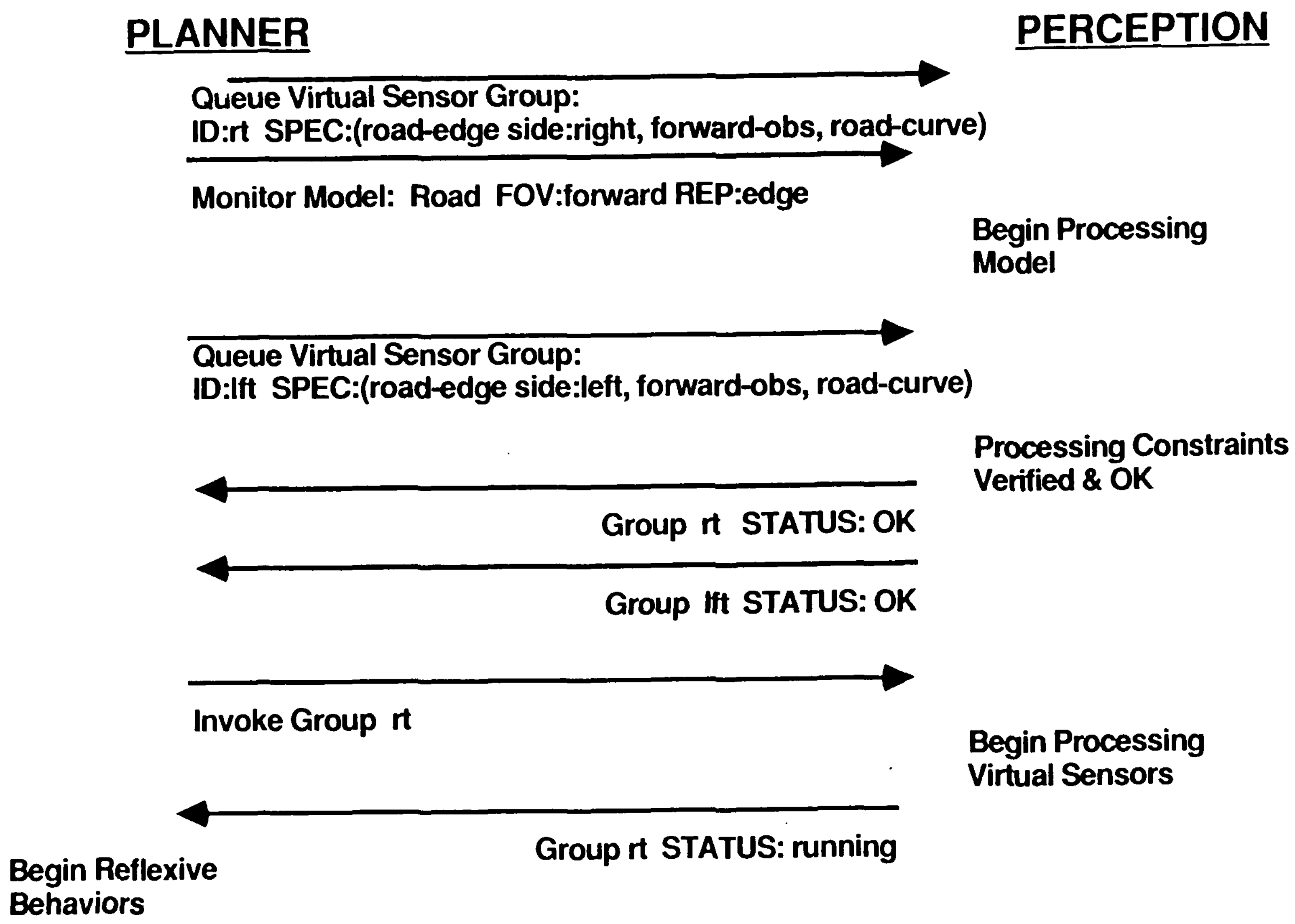
$\longleftarrow$

Road Model

Examine Model

Select Behaviors: Slow-for-curves, track-edge(rt), avoid-obs(lft)

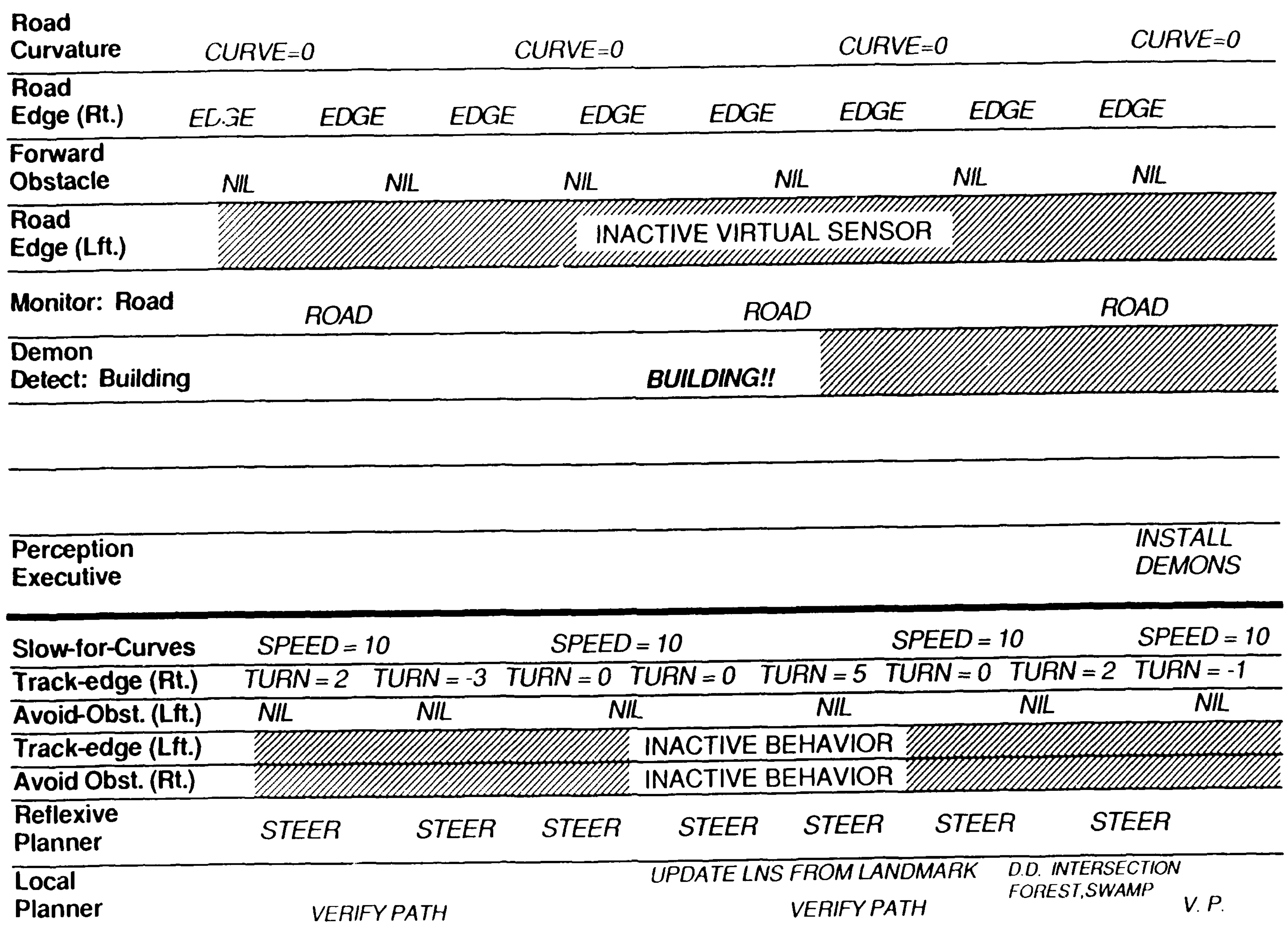
System Initialization (cont)

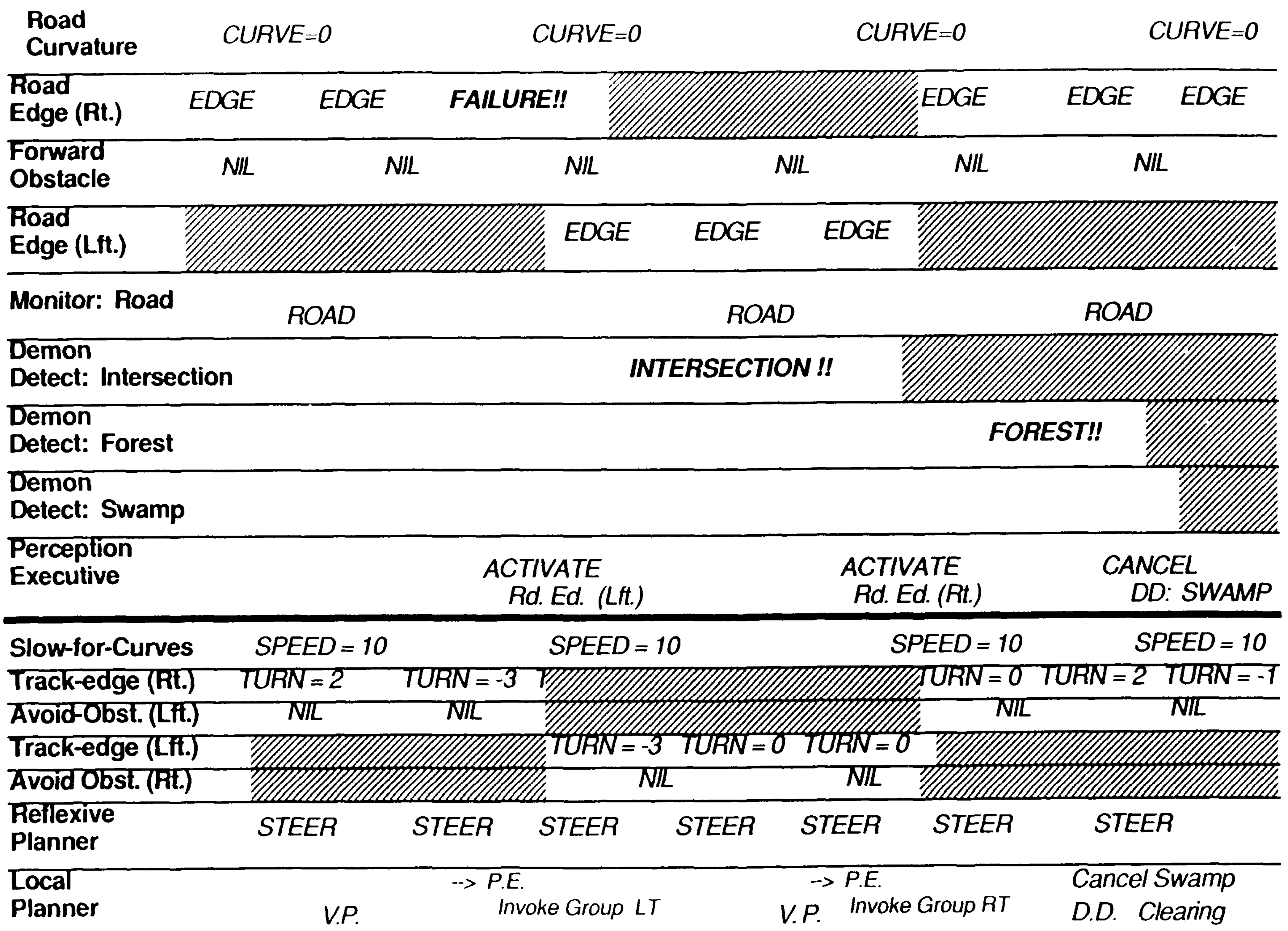

PLANNER
PERCEPTION
Queue Virtual Sensor Group:
ID:rt SPEC:(road-edge side:right, forward-obs, road-curve)
Monitor Model: Road FOV:forward REP:edge
Begin Processing Model
Queue Virtual Sensor Group:
ID:lft SPEC:(road-edge side:left, forward-obs, road-curve)
Processing Constraints Verified & OK
Group rt STATUS: OK
Group lft STATUS: OK
Invoke Group rt
Begin Processing Virtual Sensors
Group rt STATUS: running
Begin Reflexive Behaviors

DYNAMIC ROAD FOLLOWING SCENARIO

VIRTUAL SENSORS	Road Curvature	CURVE=0		CURVE=0			CURVE=0		CURVE=0
	Road Edge (Rt.)	EDGE	EDGE	EDGE	EDGE	EDGE	EDGE	EDGE	EDGE
	Forward Obstacle	NIL	NIL	NIL		NIL	NIL		NIL
	Road Edge (Lft.)	▨	▨	▨	▨	INACTIVE VIRTUAL SENSOR	▨	▨	▨
HIGH LEVEL PERCEPTION	Monitor: Road		ROAD			ROAD			ROAD
	Demon Detect: Building					BUILDING!!	▨	▨	▨
	Perception Executive								INSTALL DEMONS
BEHAVIORS	Slow-for-Curves	SPEED = 10			SPEED = 10			SPEED = 10	SPEED = 10
RIGHT	Track-edge (Rt.)	TURN = 2	TURN = -3	TURN = 0	TURN = 0	TURN = 5	TURN = 0	TURN = 2	TURN = -1
	Avoid-Obst. (Lft.)	NIL	NIL	NIL		NIL	NIL		NIL
LEFT	Track-edge (Lft.)	▨	▨	▨	▨	INACTIVE BEHAVIOR	▨	▨	▨
	Avoid Obst. (Rt.)	▨	▨	▨	▨	INACTIVE BEHAVIOR	▨	▨	▨
	Reflexive Planner	STEER	STEER	STEER	STEER	STEER	STEER	STEER	
	Local Planner		VERIFY PATH			UPDATE LNS FROM LANDMARK VERIFY PATH		D.D. INTERSECTION FOREST,SWAMP	

TIME ⟶


Road Curvature
CURVE=0
CURVE=0
CURVE=0
CURVE=0
Road Edge (Rt.)
EDGE
EDGE
EDGE
EDGE
EDGE
EDGE
EDGE
EDGE
Forward Obstacle
NIL
NIL
NIL
NIL
NIL
NIL
Road Edge (Lft.)
INACTIVE VIRTUAL SENSOR
Monitor: Road
ROAD
ROAD
ROAD
Demon Detect: Building
BUILDING!!
Perception Executive
INSTALL DEMONS
Slow-for-Curves
SPEED = 10
SPEED = 10
SPEED = 10
SPEED = 10
Track-edge (Rt.)
TURN = 2
TURN = -3
TURN = 0
TURN = 0
TURN = 5
TURN = 0
TURN = 2
TURN = -1
Avoid-Obst. (Lft.)
NIL
NIL
NIL
NIL
NIL
NIL
Track-edge (Lft.)
INACTIVE BEHAVIOR
Avoid Obst. (Rt.)
INACTIVE BEHAVIOR
Reflexive Planner
STEER
STEER
STEER
STEER
STEER
STEER
STEER
Local Planner
VERIFY PATH
UPDATE LNS FROM LANDMARK
VERIFY PATH
D.D. INTERSECTION FOREST, SWAMP
V. P.



Road Curvature
CURVE=0
CURVE=0
CURVE=0
CURVE=0
Road Edge (Rt.)
EDGE
EDGE
FAILURE!!
EDGE
EDGE
EDGE
Forward Obstacle
NIL
NIL
NIL
NIL
NIL
NIL
Road Edge (Lft.)
EDGE
EDGE
EDGE
Monitor: Road
ROAD
ROAD
ROAD
Demon Detect: Intersection
INTERSECTION !!
Demon Detect: Forest
FOREST!!
Demon Detect: Swamp
Perception Executive
ACTIVATE Rd. Ed. (Lft.)
ACTIVATE Rd. Ed. (Rt.)
CANCEL DD: SWAMP
Slow-for-Curves
SPEED = 10
SPEED = 10
SPEED = 10
SPEED = 10
Track-edge (Rt.)
TURN = 2
TURN = -3
TURN = 0
TURN = 2
TURN = -1
Avoid-Obst. (Lft.)
NIL
NIL
NIL
NIL
Track-edge (Lft.)
TURN = -3
TURN = 0
TURN = 0
Avoid Obst. (Rt.)
NIL
NIL
Reflexive Planner
STEER
STEER
STEER
STEER
STEER
STEER
STEER
Local Planner
V.P.
--> P.E. Invoke Group LT
V.P.
--> P.E. Invoke Group RT
Cancel Swamp D.D. Clearing

Road Curvature	CURVE= 3		CURVE=2			CURVE=-1		CURVE=1
Road Edge (Rt.)	EDGE	EDGE	EDGE	EDGE	EDGE	EDGE	EDGE	EDGE
Forward Obstacle	NIL	NIL	**OBS**	**OBS**	**OBS**	NIL		NIL
Road Edge (Lft.)	▨	▨	▨	▨	▨	▨	▨	▨
Monitor: Road	ROAD				ROAD			ROAD
Demon Detect: Clearing								
Perception Executive								
Slow-for-Curves	SPEED = 7		SPEED = 8			SPEED = 9		SPEED = 9
Track-edge (Rt.)	TURN = 2	TURN = -3	TURN = 0	TURN = 0	TURN = 5	TURN = 0	TURN = 2	TURN = -1
Avoid-Obst. (Lft.)	NiL	NIL	S= 3, T= -4	S= 2, T= 2	S= 3, T = 1		NIL	NIL
Track-edge (Lft.)	▨	▨	▨	▨	▨	▨	▨	▨
Avoid Obst. (Rt.)	▨	▨	▨	▨	▨	▨	▨	▨
Reflexive Planner	STEER	STEER	STEER	STEER	STEER	STEER	STEER	
Local Planner		VERIFY PATH			VERIFY PATH			VERIFY PATH

Road Curvature	CURVE=0 CURVE=0 CURVE=0 CURVE=0
Road Edge (Rt.)	EDGE EDGE EDGE **HOLD** ▨
Forward Obstacle	NIL NIL NIL NIL NIL NIL
Road Edge (Lft.)	▨ EDGE EDGE EDGE EDGE EDGE
Monitor: Road	ROAD ROAD ROAD
Demon Detect: Clearing	**CLEARING!!**
Perception Executive	ACTIVATE Rd. Ed. (Lft.)
Slow-for-Curves	SPEED = 10 SPEED = 10 SPEED = 10 SPEED = 10
Track-edge (Rt.)	TURN = 2 TURN = -3 TURN = 0
Avoid-Obst. (Lft.)	NIL NIL NIL ▨
Track-edge (Lft.)	▨ TURN = 5 TURN = 0 TURN = 2 TURN = -1
Avoid Obst. (Rt.)	▨ NIL NIL NIL NIL
Reflexive Planner	STEER STEER STEER STEER STEER STEER STEER
Local Planner	VERIFY PATH (ROAD MODEL SHOWS BAD RIGHT EDGE AHEAD) --> PE INVOKE GROUP LT VERIFY PATH VERIFY PATH **HALT** **GOAL ATTAINED**

END

10-87

DTIC